

AD-A093 680

TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CA
FAST METHODOLOGY & CASE STUDY.(U)
NOV 80

F/G 9/2

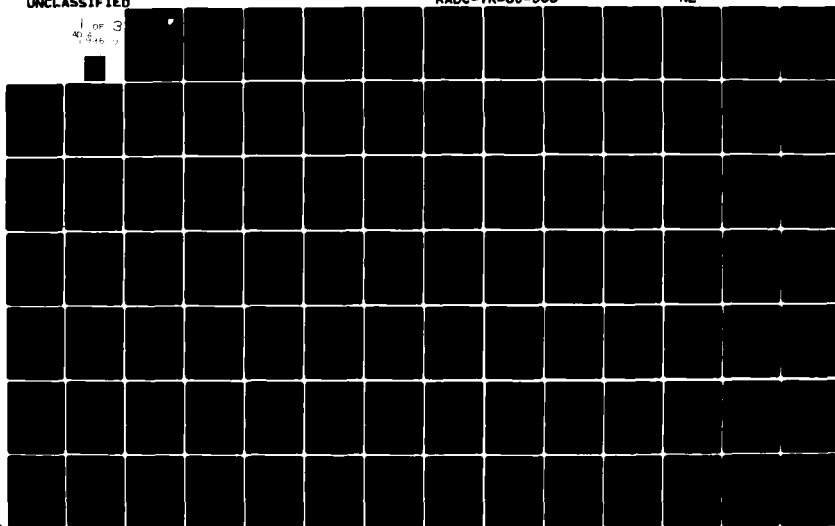
F30602-79-C-0078

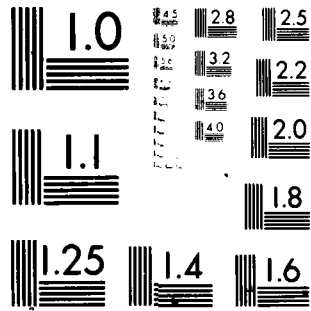
UNCLASSIFIED

RADC-TR-80-336

NL

1 OF 3
40 846 1





AD A093680

LEVEL 1



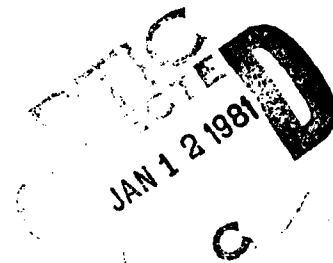
RADC-TR-80-336
Final Technical Report
November 1980



FAST Methodology & Case Study

TRW Defense and Space Systems Group

TRW Defense and Space Systems Group



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

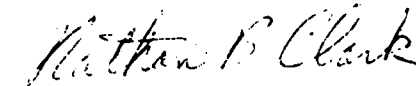
81 1 00 012

11

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

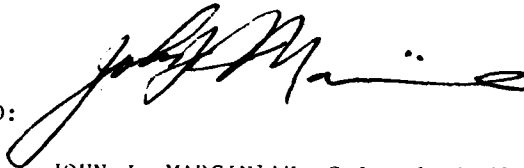
RADC-TR-80-336 has been reviewed and is approved for publication.

APPROVED:



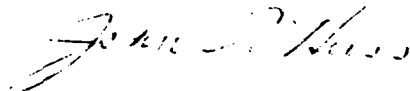
NATHAN B. CLARK, Capt, USAF
Project Engineer

APPROVED:



JOHN J. MARCINIAK, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER:



JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
18 <u>RADC</u> <u>TR-80-336</u>	2. AUTHORITATIVE STATEMENT NUMBER		
6 <u>FAST METHODOLOGY & CASE STUDY</u>	9 <u>Final Technical Report</u> <u>27 Oct 79-29 Aug 80</u>		
7. AUTHOR	10. PERFORMING ORGANIZATION NAME AND ADDRESS		
TRW Defense and Space Systems Group	11. CONTROLLING OFFICE NAME AND ADDRESS		
TRW Defense and Space Systems Group One Space Park Redondo Beach CA 90278	12. NUMBER OF PAGES		
Rome Air Development Center (ISCA) Griffiss AFB NY 13441	13. SECURITY CLASSIFICATION		
14. MONITORING AGENCY NAME AND ADDRESS (if different from Controlling Office)	15. DECLASSIFICATION DOWNGRADING SCHEDULE		
Same <u>5581, 3245</u>	UNCLASSIFIED		
16. DISTRIBUTION STATEMENT (of this Report)			
Approved for public release; distribution unlimited.			
17. DISTRIBUTION STATEMENT (of the abstract entered in 10's & 20's, if different from Report)			
Same			
18. SUPPLEMENTARY NOTES			
RADC Project Engineer: Capt Nathan B. Clark (ISCA)			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)			
Computer Architecture Hardware Software Tradeoffs Design Methodology Emulation Simulation			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)			
This report describes work done to implement and validate the Flexible, Analysis, Simulation and Test methodology developed by TRW to perform hardware software tradeoff analysis for computer system design. The methodology, partially developed with TRW IR&D funds, was tested by a case study analysis using the Defense Mapping Agency's Management Information System for Procurement (DMIS/P).			

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

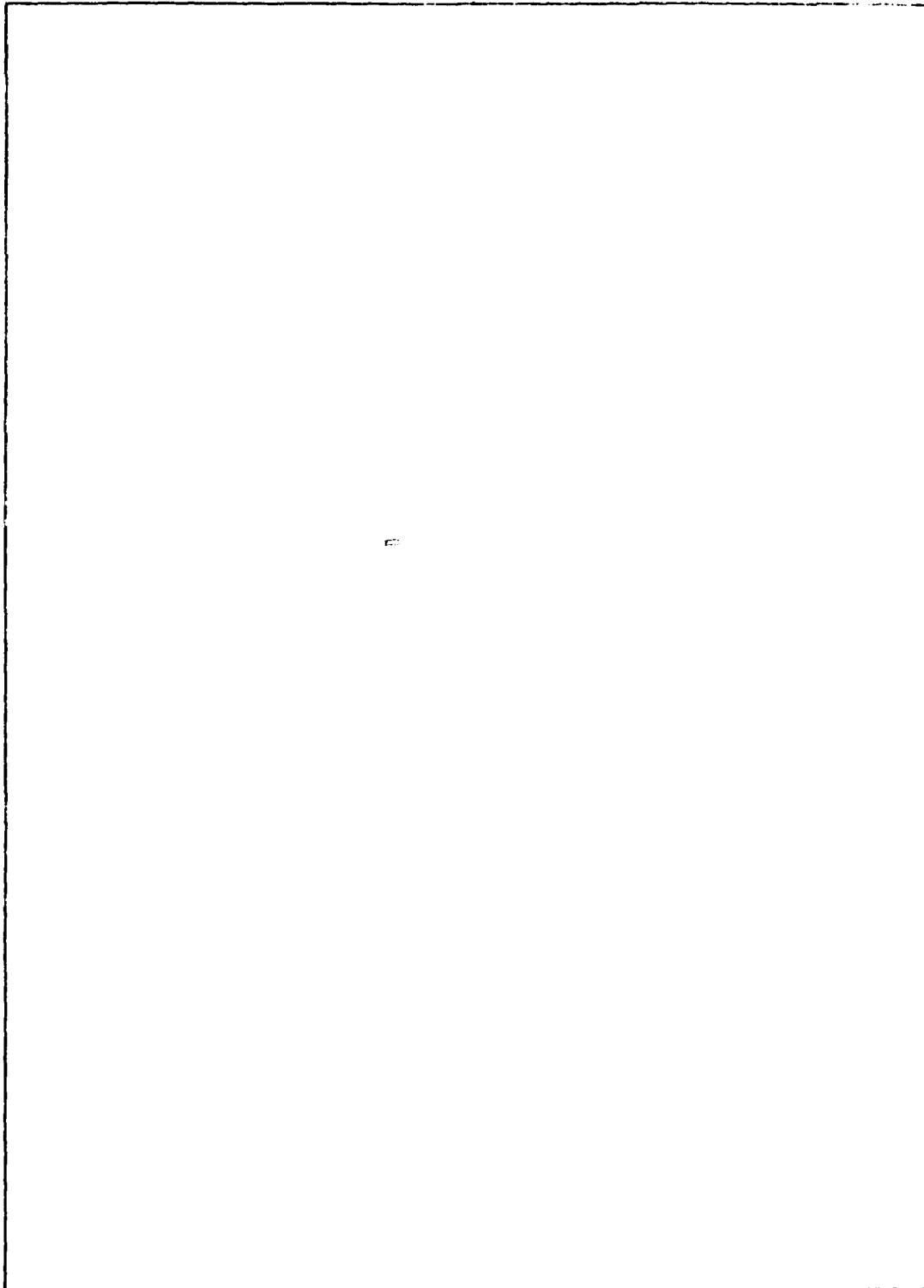
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407637

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Table of Contents

	Page
1.0 Introduction	1
2.0 Technical Background	4
2.1 Hardware/Software Tradeoffs and the Development Cycle	6
2.1.1 Total System Development (TSD) Framework Overview	7
2.1.2 Identification Phase	10
2.1.3 Conceptualization Phase	14
2.1.4 Realization Phase	17
2.1.5 Binding Phase	22
2.1.6 Implementation: Software Design Phase	27
2.1.7 Implementation: Hardware Design Phase	30
2.1.8 Manufacturing: Coding Phase	32
2.1.9 Integration Phase	32
2.2 Functional/Performance Specifications	34
2.2.1 Functional Specifications	54
2.2.2 Performance Specifications	71
2.3 Hardware/Software Methodology Fundamental	71
2.3.1 Methodological Approach to Binding	77
2.3.2 Selection of Commercial Computer Systems	79
2.3.3 Selection of Customized and Custom-Made Machines	81
2.3.4 Electing Custom-Made (VLSI) Devices	84
3.0 FAST Methodology	84
3.1 FAST Definition	84
3.2 Methodological Steps of FAST	92
3.2.1 Identification of Distribution Units	92
3.2.2 Generation of Hardware/Software Partitioning Constraints	95
3.2.3 Selection of Hardware/Software Partitioning Options	96
3.2.4 Development of Hardware/Software Partitioning Rules	97


	Page
3.2.5 Selection of Competing Candidates	97
3.2.5.1 Hardware/Software Partitioning	97
3.2.5.2 Logical Verification	98
3.2.5.3 Performance Checks	98
3.2.5.4 Qualitative Checks	98
3.2.5.5 Elimination of Incongruent Configurations	99
3.2.5.6 Binding of the Communications	99
3.2.5.7 Assignment of Cost Value Coefficients	99
3.2.5.8 Selection of Winning Candidate	100
3.2.5.9 Generation of Hardware/Software Documents	100
3.3 Discussion	101
4.0 A FAST Case Study	103
4.1 Objectives of the Case Study	106
4.2 DMAAC Organizational Model	108
4.2.1 Current DMIS/P System at DMAAC	114
4.2.2 Formal Specification of DMAAC	120
4.3 DMIS/P Design Specification	129
4.3.1 DMIS/P Informal Specifications	129
4.4 Performing Hardware/Software Tradeoffs	158
4.4.1 DMIS/P Performance Specifications	158
4.4.1.1 Relevant Performance Attributes	159
4.4.1.2 Performance Evaluation Model	160
4.4.1.3 Constraints	161
4.4.1.4 Formalization of Performance Specifications	161
4.4.2 Step by Step Application of FAST	188
4.4.2.1 Identification of Distribution Units	189
4.4.2.2 Generation of Hardware/Software Partitioning Constraints	189

	Page
4.4.2.3 Selecting Hardware/Software Partitioning Option	189
4.4.2.4 Development of Hardware/Software Partitioning Rules	190
4.4.2.5 Selection of Competing Candidate Partitions	190
4.4.2.6 Candidate Evaluation	190
4.4.2.7 Elimination of Incongruent Configurations	191
4.4.2.8 Communication Medium	192
4.4.2.9 Assignment of Cost/Value Coefficient	192
4.4.2.10 Selection of Winning Configurations	192
4.5 Discussion	193
5.0 Conclusion	199
6.0 References	203

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	<input type="checkbox"/>
By _____	
Distribution _____	
Availability _____	
Dist	<input type="checkbox"/>
f	

EVALUATION

This effort has advanced the state of the art in computer system design methodologies by implementing the FAST (Flexible Analysis, Simulation and Test) methodology for performance of hardware software tradeoffs. It also provides a case study analysis as a method of demonstrating current capabilities and areas for further implementation.


NATHAN B. CLARK, Capt, USAF
Project Engineer

Summary

This paper describes work completed under contract to RADC, F30602-79-C-0078, for the period 2 February 1979 to 29 August 1980. The goal of the project, referred to as, FAST(Flexible, Analysis, Simulation and Test); was to develop a hardware/software tradeoffs methodology. A case study was used to test the potential use of the methodology. Tools and techniques for its use are also identified.

1.0 INTRODUCTION

The design and acquisition of computer-based systems; such as weapons embedded and distributed, continues to be a difficult task. A major issue affecting the process is the need to make a hardware/software tradeoff among system architectures.

In this paper we describe a methodology intended to be used when there is a need to make decisions about what functions to allocate to hardware and what functions to allocate to software. The goal of this paper is to discuss the results of the FAST(Flexible, Analysis, Simulation, and Test), project; which was designed to define and develop the FAST methodology.

The project set out to accomplish the following specific objectives:

- (1) Define and document the FAST methodology. The methodology definition includes procedures, tools, reports and management techniques necessary for conducting computer architectural studies,
- (2) Discuss a methodology critique, which identifies potential problems and risks associated with implementing the FAST

methodology, and

- (3) Demonstrate the use of applicable portions of FAST in a selected case study.

Finally, the results are analyzed and synthesized in a case study analysis. Recommendations for future steps and additional work needed for the full development and application in DoD computer acquisition and system design activities is also given.

An underlying assumption of this effort is: A hardware/software tradeoffs analysis is a small subset of a total system design framework. Current research conducted at Rome Air Development Center and various industries have found a need for a framework from which methodologies can be developed. This framework, when fully implemented, will provide the basis for developing methodologies with capabilities for validating the design; and realizing functional elements prior to acquisition of system hardware.

Such a framework is necessary if the development of complex hardware/software systems is to be done at acceptable levels of risk. A parallel effort, of which FAST is a part, was performed to address the TSD concept. Under contract to RADC, F30602-78-C-0250 and in collaboration with TRW, a TSD framework was defined. Section 2 of this report gives an overview of the framework and the methodological steps it proposes.

The FAST definition and the methodological steps needed to perform the hardware/software tradeoffs analysis were discussed in the Interim Technical Report (see reference 27, p. 3). Section 3 of this paper summarizes the results of that portion of this project. Sections 3.1 to 3.2 presents a detailed view of FAST and explains how it relates to the TSD.

Finally, the methodology is tested in a DMA (Defense Mapping Agency) cost study. This activity is given in Section 4. Section 5 summarizes the results of the study and points out its ultimate potential.

2.0 TECHNICAL BACKGROUND

The objective of this section is to place the results described in Section 3 and 4 in proper perspective and to review the technical knowledge that established the foundation for development of the FAST Methodology and the performance of the FAST Case Study. These goals are accomplished by considering first the role played by hardware/software tradeoffs in the context of the entire system development cycle. Secondly, the basic issues and philosophy behind a general systematic approach to performing hardware/software tradeoffs are discussed. Incidental to these efforts, both notation and terminology are introduced and clarified, thus preparing the reader for subsequent sections.

The overall presentation is organized top-down, from general to particular, over several levels of abstraction. Section 2.1 considers the entire system development from a highly abstract nonprocedural point of view in order to identify the role of hardware/software tradeoffs and the way in which they relate to other system development activities. For this purpose, the concept of a methodological framework is introduced as an abstraction of the current state-of-the-art in system development methodologies.

Section 2.2 deals with issues pertinent to the functional and performance specification of systems design and the requirements they must satisfy to support systematic and potentially automatic performance of hardware/software tradeoffs. A specification language believed to meet some of the stated requirements is suggested for use in the case study.

Finally, Section 2.3 focuses on the fundamentals of hardware/software tradeoffs and proposes a general approach that all

might share. The emphasis is on establishing the complexity of the problems to be solved and the manner in which different constraints affect the nature of the potential solutions. No commitments are made to the use of one particular technique over another. Such instantiation of the general problem solving method introduced here is postponed for Section 3 where the FAST Methodology is outlined.

2.1 HARDWARE/SOFTWARE TRADEOFFS AND THE DEVELOPMENT CYCLE

Note: The reader is directed to RADC Technical Report of F30602-78-C-250 "Total System Development (TSD) Framework," author by G. C. Roman of Washington University in St. Louis, for a more in-depth coverage of this topic including an extensive reference list.

2.1.1 Total System Development (TSD) Framework Overview

Since 1968, the year when the term "software engineering" was born, significant progress has been made in understanding and controlling the software development cycle. A wealth of methodologies has evolved along with design principles and techniques, specification languages, and so on. However, the late seventies marked the emergence of two powerful trends: distribution and VLSI. As a result, the software/hardware relationship has grown more complex while, at the same time, systems development has begun to demand a greater understanding of hardware architectures and the impact they might have on a system's design and characteristics.

A critical point has been reached where software development and hardware selection and design must be brought together under the umbrella of a unifying conceptual framework. Total System Development (TSD) is put forth as a candidate framework particularly well suited for this task. TSD is meant to establish the basis for an integrated approach to computer systems development and to contribute to the development of new methodologies which treat systems as distributed software/hardware aggregates, thus breaking the barrier between software and hardware design.

As suggested earlier, TSD is based on the analysis of current software system methodologies, hardware selection methods, and hardware design approaches. TSD phases have been selected by grouping together system development activities related to each other and sharing the use of a common knowledge base (theory, techniques, skills, etc.), a variation on the principle of separation of concerns. As a result, some phases deal with issues of the application domain, others involve computational structures, etc. Figure 2.1.1.-1 contains a diagram depicting the TSD phases and their fundamental interrelationships.

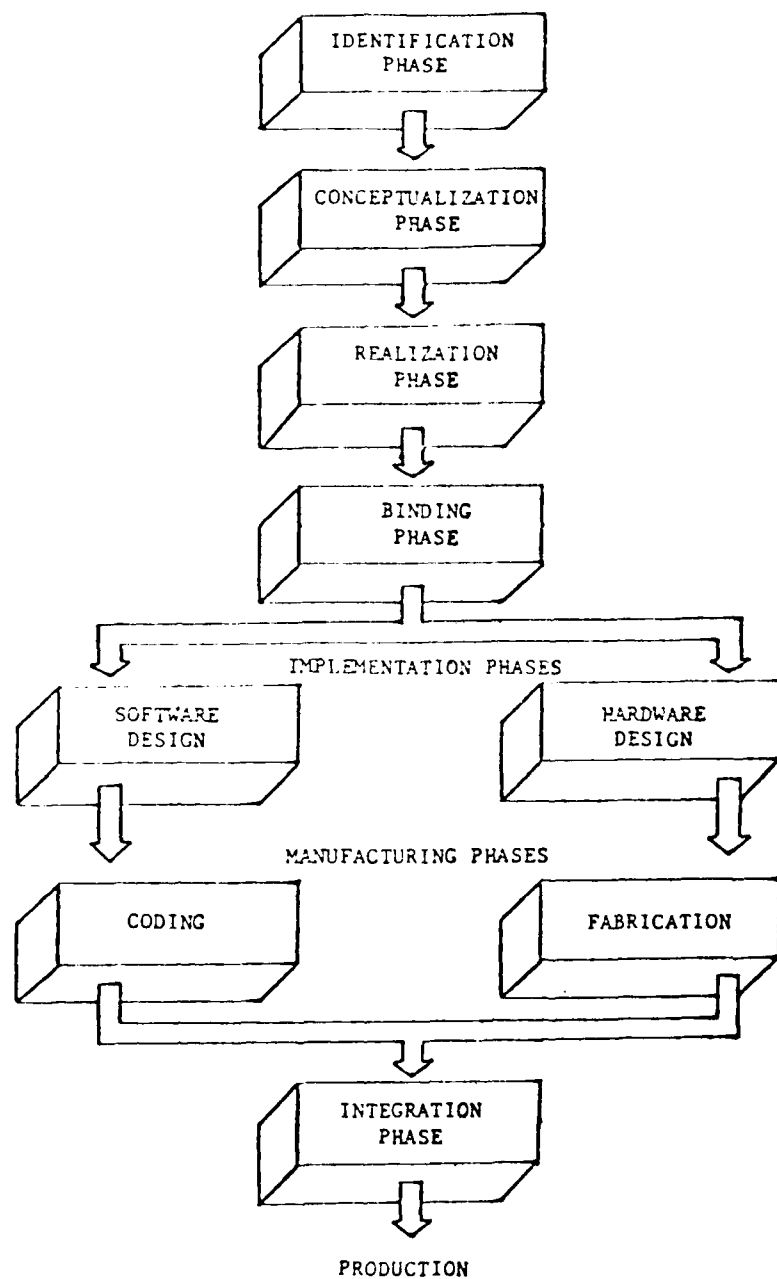


Figure 2.1.1.-1: Total System Development(TSD) Framework

Short descriptions of each phase are included in the remainder of this section. A phase by phase detailed presentation follows starting with the next section.

The problem IDENTIFICATION phase is informal in nature and has an exploratory flavor. It denotes the gathering of information about those activities of the application domain which need to be supported by some proposed system. A good understanding of the application domain is essential for the successful completion of this phase.

During the CONCEPTUALIZATION phase, the information previously gathered is organized as a formal model called a conceptual model. The entire phase is application domain dependent and rests on the ability to formalize the problem domain. The conceptual model becomes the basis for the rest of the development process and, as such, there are good reasons to propose it also as a foundation for all contractual agreements.

The system REALIZATION phase moves the development process from the application domain modeling into the computational domain. For the first time, a realization of the system is proposed in terms of processes, data, communication, and abstract processors (idealized machines) to which data and processes are bound. Performance evaluation plays an important role during this phase by helping in the selection between alternate realizations. It ought to be stressed at this point that the realization phase results in a technology independent solution to the problem at hand. Considering the rapid technological changes which have occurred during the last decade and continue to be manifest today, it becomes necessary to separate the technology independent aspects of the design from decisions which are technology dependent. The criterion used in this document naturally separates these phases which in many methodologies today are not yet differentiated.

In contrast with the realization phase, BINDING is a heavily technology dependent phase. It recognizes the unique characteristics of hardware and software, but also the hardware/software duality. The role of the binding phase is that of assigning processors to specific hardware and, during this process, to optimize the system based on performing hardware/software trade-offs. This process may take a variety of forms from vendor selection (based on commercially available hardware/software) to choosing custom hardware and software (in which case software and hardware requirements need to be specified). During the same phase, the characteristics of the communication medium are also bound.

IMPLEMENTATION denotes two phases: SOFTWARE DESIGN and HARDWARE DESIGN. They both deal with the selection of a particular implementation (composition of functions or building blocks) for custom built components. At the present time there exist significant differences between hardware and software design procedures (despite the duality principle), but it is conceivable that in the future the two phases may resemble each other more and more as the two knowledge bases converge.

Because of the software/hardware differences, MANUFACTURING denotes two phases corresponding to two technological domains: CODING, which refers to the process of materializing a particular software implementation through the use of some programming language, and FABRICATION, which is the process by which hardware components are built based on solid state and electronics technologies.

The INTEGRATION phase deals with the assembly of the system from component parts and the associated testing procedures. Subsequent to integration the system is installed and put into production.

2.1.2 Identification Phase

Identification is the first phase of the TSD framework. It encompasses activities whose goal is that of establishing the groundwork required to start system development. This phase attempts to identify why a system should be built, what it is supposed to do, how it relates to the production environment, and what the constraints to be met by the system are. The identification phase has a highly informal character (formalization requires first an understanding of the application domain, and requires careful consideration of the human factors involved in the process of creating a good communication link between the developer and the customer.

The output of the identification phase is a report detailing, in some organized but informal way, the results of the developer's explorations of the customer's world. It corresponds, in some sense, to the experimental or field data used in disciplines like physics or anthropology. As such, the viewpoint tends to be local, the level of abstraction is low, and the correlations are few. The completeness and accuracy of the data are the key issues facing the writer of the report. The fundamental steps involved in the identification phase are:

- (a) Exploration
- (b) Report generation
- (c) Report evaluation

The exploration step is meant to accomplish two highly interrelated tasks: to create a communication link between developer and customer, and, to establish the role played by the proposed system. The report generation step must produce a complete and unambiguous description of

the issues revealed during the exploration step. The report evaluation is an analytical step which attempts to determine the successful completion of the identification phase.

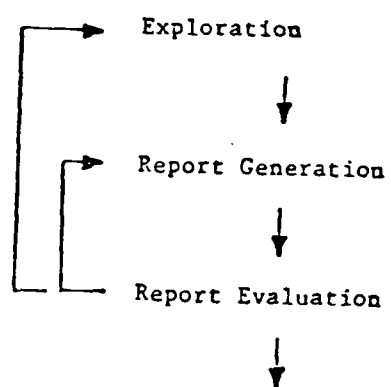


Figure 2.1.2-1: Identification Phase

Typically, the identification phase of most system development methodologies also includes steps which, while technically non-essential, have great practical significance. They tend to be oriented toward the generation of preliminary studies regarding anticipated benefits or market value, cost estimation, development time and needed resources, potential environmental, social, and legal implications, etc. Such investigative work is important in assisting decision making processes both in the developer and customer organizations. Based on these studies, unwise system development may be stopped before committing too much money and resources. However, one must emphasize that the informal nature of the identification phase can hardly support a formal study of complicated issues such as those mentioned above. More definitive studies become possible only in the conceptualization phase.

2.1.3 Conceptualization Phase

The information obtained in the identification phase is formalized during the conceptualization phase. The result is a conceptual model which formally defines all aspects of the application domain to be supported by the system under development. The conceptual model is intended to play a multitude of roles: it defines the system's functionality, boundaries, and interfaces to the application environment; it forms the basis for analytical studies of a technical (e.g., cost prediction) type; it established a firm foundation for all contractual agreements; and it helps in understanding the customer's problem representing the most important communication link between developer and customer. Informal models can rarely satisfy all these needs. The conceptual model must be precisely formulated in order to provide concise means of expression and powerful notation. Occasionally more than one conceptual model may be required for a complete description of the problem.

Jumping directly from identification to realization is an important factor in the failure of many systems being developed today. Informal descriptions of the problems tend to be incomplete, self-contradictory, hard to verify, and ambiguous. Furthermore, many fundamental problems are obscured by the verbosity of the reports and the simplicity of the examples. Such issues can be resolved only by systematic formal modeling and analyses.

The steps comprising the conceptualization phase are as follows:

- (a) Formalism selection
- (b) Formalism validation

- (c) Conceptual model construction
- (d) Conceptual model verification
- (e) System boundaries selection

The conceptualization phase starts with the selection of a formalism able to represent the problem. The formalism may be newly developed or already available from some source. In either case, the selection needs to be validated as being suitable for the task before too much effort is invested in using it. Subsequent to validation, a conceptual model of the problem is built and later made subject to systematic verification by both developer and customer. Finally, the conceptual model may be used to select, in agreement with the customer, the boundaries of the system to be developed.

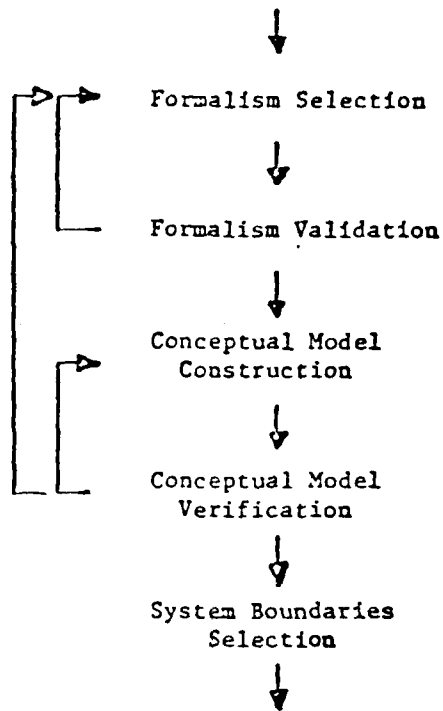


Figure 2.1.3.1: Conceptualization Phase

2.1.4 Realization Phase

System design activities start with the realization phase whose main function is that of generating a technology independent system specification called the processing model. Technology independence is achieved by maintaining the design activities at a very high level of abstraction. The processing model is conceived as the result of a highly interactive sequence of synthetic and analytic steps. The synthetic steps represent design activities aimed either at altering an unsatisfactory design solution or at adding new details to the model. The analytic steps establish, on one hand, the logical correctness of the design and, on the other hand, conformity with given qualitative and quantitative constraints which may have originated with the customer or developer, or may represent generally accepted rules of the trade.

The processing model is defined in terms of data, messages, events, processes, and processors. Data and messages are treated as primitive entities. Events are defined as relevant changes in the state of the system or its environment. The events materialize in terms of data value modifications and message receiving or sending. Disjoint sequences of events form processes. Both data and processes are partitioned into classes. They establish units of distribution within the system called processors. Processes are restricted to accessing only data assigned to the same processor, but they may exchange messages with processes bound to other processors.

The motivation behind developing the processing model is three-fold. First, the model describes top level system design from the point of view of functionality and behavior at an abstract enough level so as to be independent of detailed technical considerations. Secondly, theoretical solution to the data and process distribution problem and communication protocol selection are proposed. Thirdly, the processing

model is envisioned to become the basis for both hardware/software partitioning and hardware selection, which take place in the binding phase. As such, in addition to the components described above, the processing model should include performance related information such as data volume, message rates, data access patterns, etc. This information could originate, in part, in the conceptual model and among customer provided constraints while the rest would have to be generated during the performance checks to which the model is subjected in the realization phase.

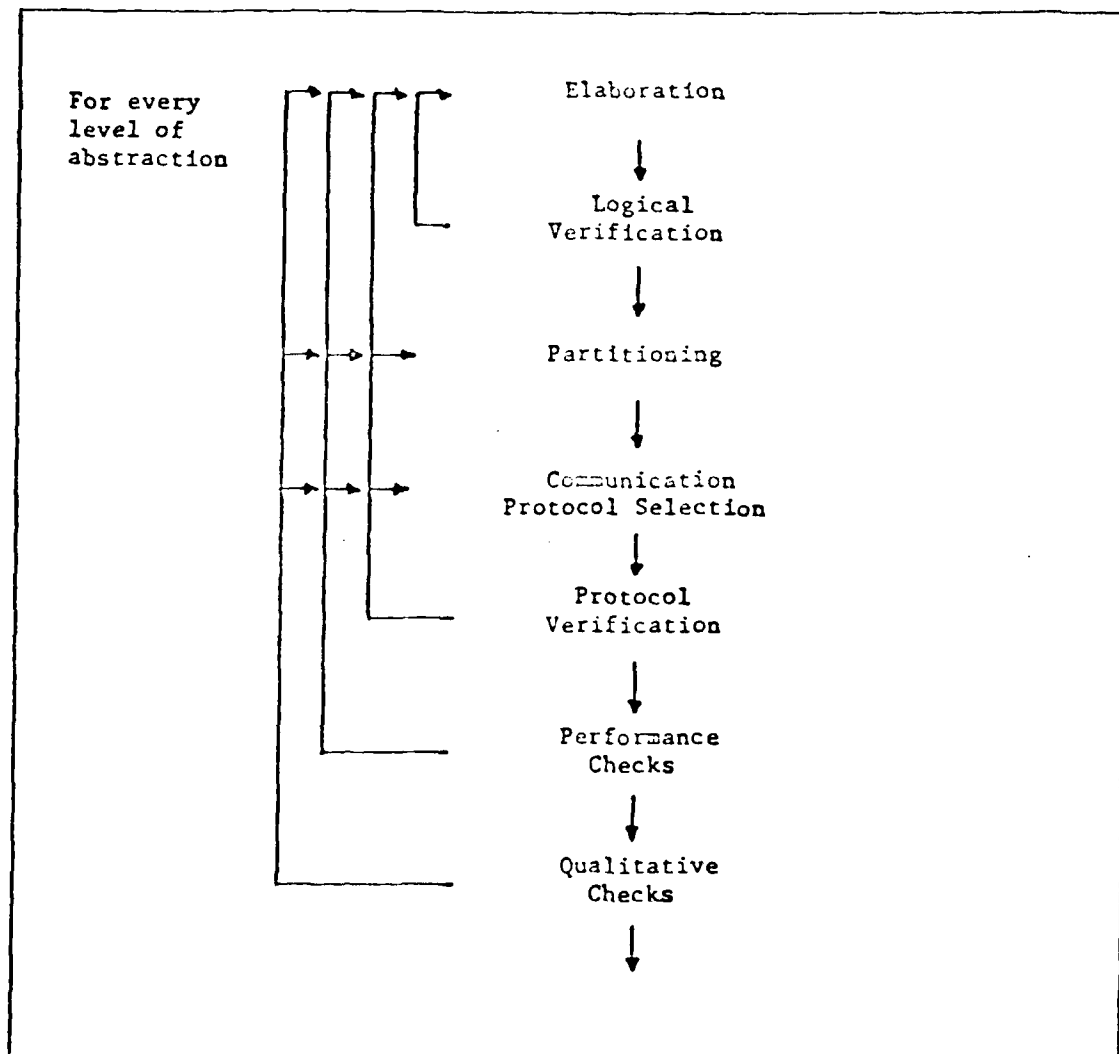


Figure 2.1.4-1: Realization Phase

The method employed in developing the processing model is outlined below. (While trouble at one level may require redefinition of previous levels, such feedback loops have been omitted for the sake of simplicity.) The proposal is to separate the realization phase into seven basic steps to be repeated a number of times, as necessary. These steps are:

- (a) Elaboration
- (b) Logical verification
- (c) Partitioning
- (d) Communication protocol selection
- (e) Protocol verification
- (f) Performance check
- (g) Qualitative check

The strategy is to start with an initial nondistributed solution and to propose an acceptable distributed system by considering both explicit and implicit customer originated constraints. There are minimum distribution requirements, performance characteristics, survivability, fault tolerance, reliability level, etc.

The elaboration step proceeds by establishing the set of events relevant to the particular level of abstraction and group them into parallel processes. A shared memory communication model is assumed at this point. The resulting specification is then subject to logical verification, which attempts to determine its consistency with respect

to the level above (or below) and with respect to the conceptual model. During partitioning, processors are approximated in terms of the data and the processes assigned to them. This is achieved by continuing to employ the shared memory model while attaching different costs to accessing data within the same processor and across processors. During the next step, the communication protocol selection step, the model is modified so as to include message passing and to restrict processes from directly accessing data from a different processor. These activities are followed by a new logical verification in the protocol verification step. Subsequently, performance and qualitative checks are carried out in order to evaluate the design's conformity to the set of realization pertinent constraints.

Because of the size of the systems and the complexity of the decisions involved in the realization phase, the same seven steps would have to be repeated for each successive level of abstraction employed in the processing model. Consequently, as the need for further distribution is determined, one or more processors on level n may be further distributed by employing the same procedure, thus generating level $(n+1)$. As an example, processors on level one may represent network node while on level two they may abstract single machines. The level of detail at which the refinement and distribution activities (i.e., the realization phase) are curtailed is a function of the ability to assess compliance with all relevant constraints, however, failures in being able to carry out the binding may require later resumption of this phase.

2.1.5 Binding Phase

Determination of the hardware/software boundary and selection of particular hardware components of the system are the goals of the binding phase. All decisions taken during this phase are controlled by market availability, technological state-of-the-art, and manufacturing capabilities. The constraints imposed by these factors are manifest not only in terms of sheer availability of hardware having certain given qualities, but also in the cost of various existing components. Binding and all subsequent phases are heavily technology dependent and are expected to change as long as technological progress takes place. As the cost structure of alternate options varies with time, the binding techniques may change substantially. Such a phenomenon is actually observable today in a trend away from the large mainframes and toward distributed systems.

Because binding is what hardware/software tradeoffs (i.e. FAST) are all about a comprehensive treatment of this phase is delayed until Section 2.3.1 where a methodological approach to binding is proposed.

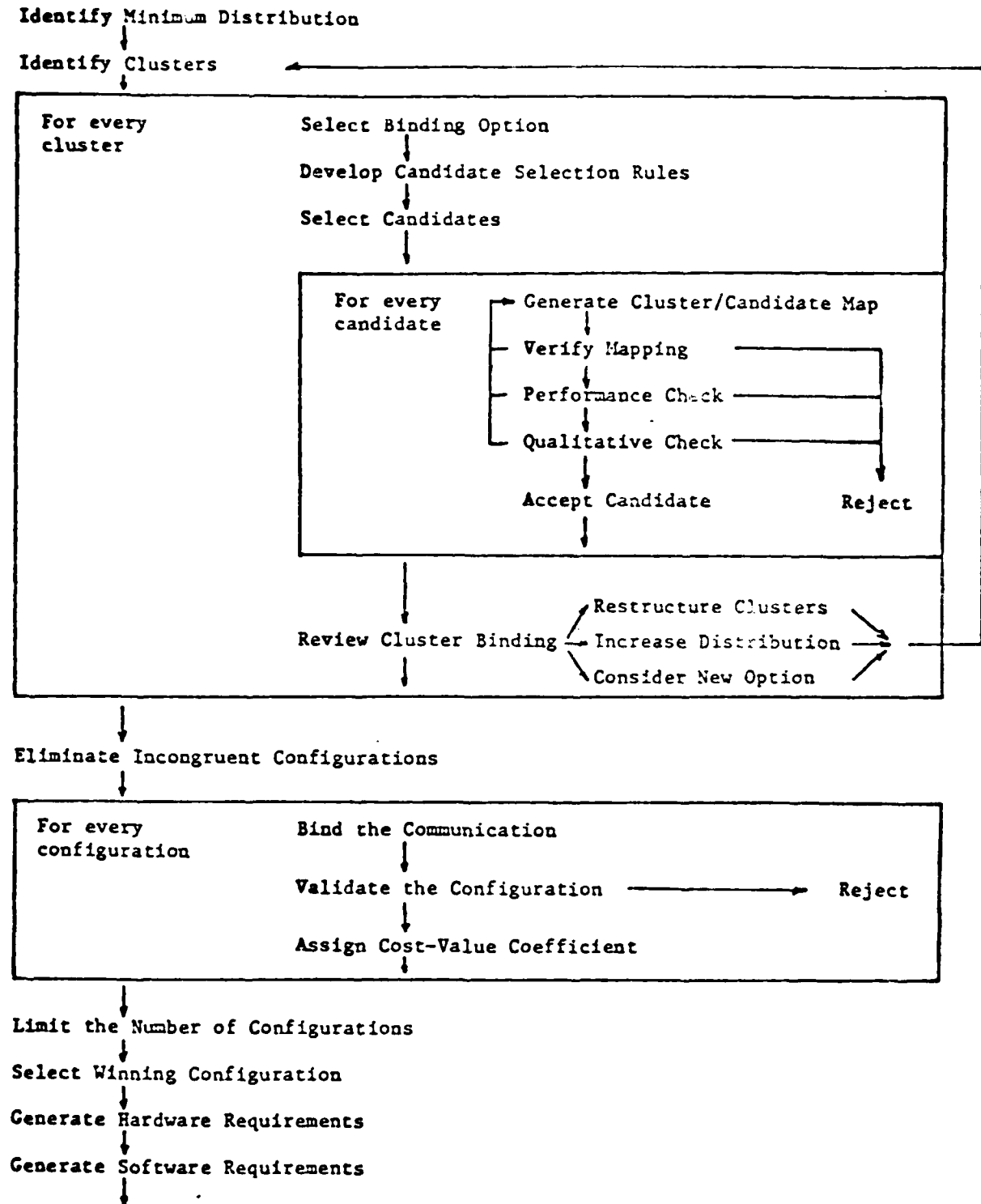


FIGURE 2.1.5-1: Binding Phase

2.1.6 Implementation: Software Design Phase

Each software system is specified by the corresponding requirements definition generated during the binding phase. The requirements definition specifies the functionality of the software system, the data it controls, its interfaces to other software systems, the programming language to be used during coding, all performance constraints to be met by the software system, and the machine characteristics needed to predict compliance with the stated constraints. The role of the software design phase is to select, based on the requirements definition, an effective software system design. The software system design specification should identify all software modules, the inter-module interfaces and the internal module design.

The software design phase follows. design phase follows. The first step deals with the selection of an overall software system architecture defined as a collection of modules that cooperate in performing the various functions of the software system. Once an overall architecture is chosen, the detail design of the module interfaces must be carried out in order to enable the logical verification of the software system and the unambiguous design of the component modules. The interfaces may vary considerably in nature. They may take the form of in-core data-tables, files shared or passed between modules, or a common database.

The design of the individual software modules is preceded by the verification of the software architecture and an evaluation of potential performance problems. The verification typically entails a convincing demonstration of the well-formedness and self-consistency of the specification, and its consistency with respect to the requirements definition.

This activity is not fundamentally different from the corresponding step of the realization phase. The performance check requires software modeling or simulation in order to evaluate the software design against the performance constraints. Qualitative checks of a nature similar to those employed during the realization phase are also recommended.

The issues related to individual module design are relatively mundane (today), and the amount of literature written on this subject is excessively large. It deals primarily with the principles of structured programming.

Upon completion of the module designs, the entire software system design may require reverification at a more detailed level and a reevaluation of the performance check due to the availability of additional information about the processing done by individual modules.

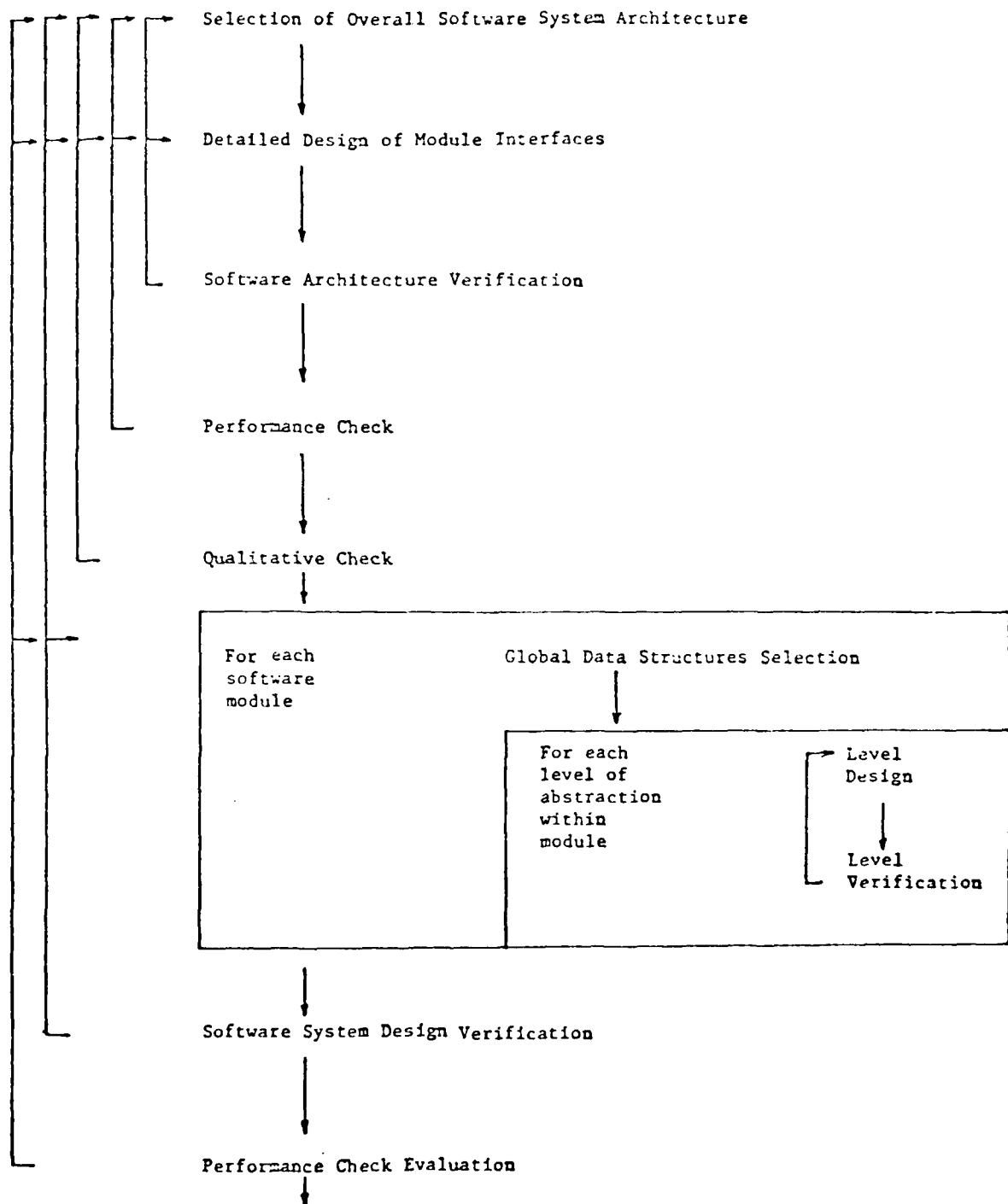


Figure 2.1.6-1: Software Design Phase

2.1.7 Implementation: Hardware Design Phase

During the binding phase, special hardware devices or custom machines may be identified as being needed in order to meet special performance requirements. On such occasions, the output of the binding phase would have to include the requirements definition of each custom-made hardware unit. The requirements ought to provide a functional specification of the unit, precise description of all hardware interfaces, storage requirements, and all pertinent constraints (performance, manufacturing, reliability, fault-tolerance, etc.).

The design of each hardware unit proceeds with the generation of a hierarchy of design specifications, each representing a level closer to the bottom line technology and each requiring a different specification language. Five conceptual levels for describing, understanding, analyzing, and designing hardware have been predicted. The levels are:

- Macro level, also called PMS (processor-memory-switch). It consists of a macro view of the architecture of the hardware unit.
- Register Transfer level. It includes the specification of registers and functional transfers such as logical and arithmetic operations. Register Transfer languages were among the first to be developed.
- Switching Circuit level. It is separated into sequential and combinatorial logic circuits. Switching theory forms the basis for formulations at this level.

00663 - Electrical Circuit level. It brings the design effort into the realm of electrical engineering by dealing with

electrical diagrams; their components are resistors, inductors, capacitors, voltage sources, nonlinear devices, etc.

- Solid State level. It deals with the realization of the electrical circuitry on single chips of semiconductor material as chip layouts.

Upon completion of each level, a series of verification steps need to be performed to assure the fact that the design meets the requirements. Simulations based on the specification generated at the particular level play a major part in the evaluation process. By means of simulation the unit is "tested" long before any manufacturing starts. The high cost of prototypes makes errors unacceptable. Some simulation may be utilized in establishing compliance with the performance constraints, sensitivity to faults, etc.

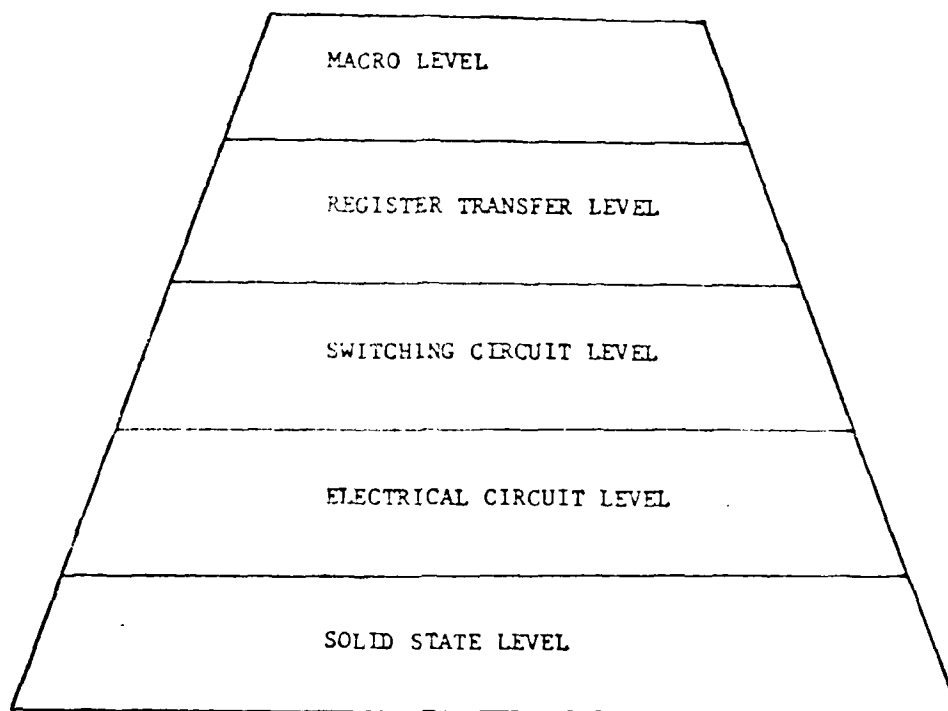


Figure 2.1.7-1: Hardware Design Hierarchy

2.1.8 Manufacturing: Coding Phase

Module designs materialize into programs during coding phase. Three interrelated activities cooperate in the generation of each program: code development, verification, and testing. These activities are often carried out in this exact order, but an effective way to proceed during the coding phase is to integrate the three activities and perform them concurrently rather than sequentially.

Code development strategies fall into several basic categories: top-down, bottom-up, and iterative growth. Since the module design already exists and is assumed correct, the code development strategy selection could be viewed as being arbitrary. However, because verification and testing need to be carried out concurrently, it is advisable to select that strategy which minimizes the testing effort. As a new program section is added, its correctness ought to be established and a new series of tests performed. Although this is apparently a time-consuming approach, experience has shown it to be much safer and more economical than the debugging of programs that have been coded in their entirety first.

Additional systematic verification and testing is required at the time of program completion. Some tools are available to aid in testing and debugging. With respect to the final verification, one has to establish compliance with programming standards, correctness with respect to the module design, effective coding of module data structures, proper trade-off between efficiency and maintainability, etc.

2.1.9 Manufacturing: Fabrication Phase

This title is included only as a reminder to the reader. The

activities involved in manufacturing chips, circuit boards, devices,
etc. is outside the scope of this undertaking.

2.1.10 Integration Phase

The integration phase includes all activities related to coupling together separately developed hardware and/or software components. The strategies employed are heavily system dependent, but the success (i.e., low number of coupling errors) rests on the quality of the interfaces' design. In other words, integration is trivial if no design or manufacturing errors occurred. As such, all efforts need to be concentrated in the other phases to make sure that easy integration is possible. Errors discovered during the testing procedures of the integration phase may trigger very expensive backtracking.

An elaboration of the integration phase is not needed for the purpose of this report.

2.2 FUNCTIONAL/PERFORMANCE SPECIFICATIONS

The hardware/software tradeoffs analysis carried out during the binding phase (Section 2.3) presupposes the existence of a functional/performance specification of the system, i.e., the processing model generated by the realization phase. This section illustrates one way in which the processing model could be described, in terms of a formal functional/performance specification language. While the language development was motivated primarily by the needs of the case study, it also offered an excellent opportunity to investigate:

- o The nature of the input to the binding phase,
- o The relationship between functional and performance specifications,
- o Ways of extending existing specification languages (such as RSL) to better support hardware/software tradeoffs analysis.

2.2.1 Functional Specifications

The approach chosen for describing distributed systems is based on a hierarchical structure where a process at one level is refined, on the next level, as a net of communicating non-deterministic processes.

2.2.1.1 Process Definition

The definition of a process has been influenced by earlier work on the specification of abstract data types by Guttag [12], on the use of abstraction mechanisms in CLU (Liskov and al [4]) and Alphard (Wolf and al [5]), on the verification of hierarchical specifications using SRI modules (Robinson and Levitt [2]), and on event ordering (Greif [10]). Each process is described by the data entities it controls and their invariant properties, a set of procedures acting upon the data above, message sending and receiving procedures associated with appropriate input and output ports, and a set of rules controlling the process behavior.

DEFINITION:

A process p is defined as

$$p = (D(p), T(p), R(p), S(p), B(p))$$

where

$$D(p) = (Q(p), H(p))$$

with $Q(p)$ denoting a set of entities controlled by p and $H(p)$ representing a predicate called the data invariant. It describes the invariant properties of $Q(p)$. The fact that

$H(p)$ is satisfied or not may be determined strictly by examining the entities in $Q(p)$. $D(p)$ is called a simple data abstraction.

$$T(p) = \{t(p_i) :: (A'(p_i), A''(p_i), V(p_i)) \text{ for } i=1, \dots\}$$

defines a set of transformational procedures over the data described by $D(p)$. Each procedure is given in terms of three predicates: an input assertion and an output assertion over the data entities in $Q(p)$ and a set of supplied parameters, and a third predicate describing the value returned by the procedure.

$$R(p) = \{r(p_i) :: (\text{true}, \text{true}, V(p_i)) \text{ for } i=1, \dots\}$$

identifies a set of message receiving procedures. $V(p_i)$ establishes the required properties of the incoming message which is made available as the return value of the procedure.

$$S(p) = \{s(p_i) :: (A'(p_i), A'(p_i), \text{true}) \text{ for } i=1, \dots\}$$

identifies a set of message sending procedures.

$$B(p) \leq (T(p) \cup R(p) \cup S(p))^*$$

represents the process behavior given in terms of possible sequence of events. Any procedure invocation is treated as a single indivisible event. The manner in which $B(p)$ is specified is discussed later.

The notion of a simple data abstraction, $D(p)$, is new only in name but not in essence. It is present in Alper [5] where the data invariant $H(p)$ is called an "abstract invariant" and may often be seen

in the database literature taking the form of "internal consistency criteria". The key about the definition of $H(p)$ is the fact that its validity may be checked statically, i.e., regardless of the history or the events (procedure invocations) that might have occurred. This particular attribute may help simplify process analysis by contributing to breaking proofs into simpler, more manageable steps.

The transformational procedures are procedural abstractions similar to those employed in the definition of abstract data types [12]. Assertions have been suggested for the description of the transformational procedures due to their nonprocedural quality and because they are needed in later proofs. Nevertheless, equivalent operational specifications may be preferred under certain circumstances.

For reason of unity and uniformity, message sending and receiving are treated as procedure invocations. Neither sending nor receiving procedures affect the data entities under the process' control. Sending procedures accept as a parameter a value to be sent, while the receiving procedures return a value each time they are being invoked. The semantics of the communication is not part of the process definition but they are included in the description of the net to which the process belongs, if any. This is why, as far as the process is concerned sending and receiving are simple local operations.

The description of a process is incomplete if its behavior, $B(p)$, is not specified. In sequential programming the behavior is given by the flow of control defined by means of predetermined control abstractions called constructs (IF-THEN-ELSE, WHILE-DO, etc.). Concurrency, however, requires more sophisticated approaches. The most straightforward approach is to assume each process to be a sequential program (thus using the ordinary flow of control constructs) but to augment it with several communication primitives (e.g., send, receive) whose semantics establish the rules for interaction between concurrent

processes. PLITS [14] and Hoare's proposal for communicating parallel processes [15] are relevant examples falling in this category. Hoare's ideas are particularly important because of the inclusion of nondeterminism (guarded commands) which, in our opinion, is fundamental to the ability of modeling distributed systems. One problem regarding this particular approach to behavior specification is its highly procedural nature which makes it inadequate particularly for use in hierarchical specifications.

Less procedural methods have been proposed: path expressions controlling access to monitors in Path-Pascal [6], event expressions specifying the desirable behavior in DREAM [7], flow expression [16], event ordering [10], etc. Among them Greif's partial orders over events has been judged to be the most appropriate because of its power of expression, conceptual simplicity and non-procedural nature. A version thereof has been adopted for the purpose of describing the process behavior and, as direct consequence, the behavioral constraints over the net resulting from the refinement of the process. The process behavior $B(p)$ is specified as a set of expressions called behavior formulas defined below.

DEFINITION:

A behavior formula is a sentence generated through the use of the following rules:

```

<behavior-formula> ::= <behavior-formula> ^ <behavior-formula>
                  ::= <behavior-formula> ?? <behavior-formula>
                  ::= ~ <behavior-formula>
                  ::= <quantifier> <event-var> : <behavior-formula>
                  ::= <event-order>
                  ::= <event-var> = <event>

<event-order>    ::= <event> < <event>

```

```

::= <event> << <event>
::= (<condition>) <event>
::= (<condition>) <event> < <event>
::= (<condition>) <event> << <event>

<condition> ::= <condition> and <condition>
::= <condition> or <condition>
::= ~ <condition>
::= <quantifier> <value> : <condition>
::= <message-receiving-procedure>?
::= <predicate-over-value-designators>

<event> ::= <list-of-new-value-designators> =
<transformational-procedure>
(<list-of-value-designators>)
::= <message-receiving-procedure> ?
    <list-of-new-value-designators>
::= <message-sending-procedure> !
    <list-of-value-designators>

<value designator> ::= <letter> <string-of-letters-and-numbers>

```

Events correspond to instances of procedure invocations. Their returned values receive unique names, value designators, to be used for further reference in expressing conditions or as parameters to other procedures. Here are three sample events:

```

z = TRANSFORMATION (x,y)
  RECEIVE ? u
  SEND! (v,w)

```

Conditions such as

```

(x=y and RECEIVE?)

```

are predicates over returned value but also allow for determining the

existence of potential messages pending at a particular port. They play a role similar to the conditions of a SELECT construct. For instance, the formula

$$(x=y \text{ and } \text{RECEIVE?}) \text{ RECEIVE ? } z$$

indicates that some message z is read in whenever x and y denote the same value and there is a message pending on the receiving port.

Event ordering is denoted by the symbol '<' which when placed between two events, $E1$ and $E2$, ought to be understood to mean "event $E1$ is followed immediately by event $E2$ ". A formula such as $E1 < E2$, however, should be read as $E1$ is followed by $E2$ ". Conditions preceding an event ordering apply to the ordering and not to the event present on the left hand side of the order symbol. Therefore,

$$(x=y) E1$$

means $E1$ may occur if x and y denote the same value, while

$$(x=y) E1 < E2$$

states that $E2$ immediately follows $E1$ whenever $E1$ occurs but only if x and y denote the same value.

The description of the process behavior most often involves many formulas. Value designators used across formulas are assumed to refer to the same value. Consequently attention must be paid to the fact that the use of a particular value, under all circumstances, ought to be preceded by its generation. With these short explanations, the meaning of the behavior formulas ought to be easily grasped. On occasion short forms of the formulas will be used. For this reason the reader needs to keep in mind the following simple conventions: (1) value designators are

omitted where superflows, (2) all procedure names appear in capital letters and on occasion may include lower case subscripts, (3) all value designators use lower case letters, (4) formulas of the type $E1 < E2$ and $E2 < E3$ may be combined giving $E1 < E2 < E3$, and (5) superscripts denote distinct instances of the same entity. Using these rules, one version of the reader/writers problem for a process that accepts read and write requests from k other processes may be expressed by means of three behavior formulas:

$V_{i,j}$ with $1 \leq i, j \leq k$

```
READ.REQ(i)! READ.REQ(i) < BEGIN.READ(i) << READ(?) <<...<<
Read(?) << SEND.DATA(i)
```

```
WRITE.REQ(j)! WRITE.REQ(j) < BEGIN.WRITE(j) < WRITE(?) <...<
WRITE(?) <END.WRITE(j)
```

```
?WRITE(j) (BEGIN.READ(i) << WRITE(j) <<SEND.DATA(i))
```

The design of a device performing real-time hidden surface elimination will be used for exemplification purposes. This example illustrates the basic ideas on a real life sophisticated problem involving very severe and also diverse performance constraints. The example is separated into several parts. The first part deals with the functional description of a proposed real-time hidden surface elimination device [17]. The device assumes objects to be represented as selections of colored planar triangles in some three-dimensional environment, a cube of some predetermined size $2k$. Objects may be subject to various real-time transformations (e.g., rotation, scaling, translation). The screen is assumed to contain $m \times n$ pixels, i.e., image elements. The image appearing on the screen is that seen by an observer through a window of size $m \times n$ placed on the center of one of the environment cube's faces. For simplicity only, the viewpoint is

assumed fixed at infinity (orthonormal view).

The device is presumed to contain a description of all the objects in the environment. Furthermore, it has three ports: one for receiving commands specifying object transformations (GETCMND); the second for receiving signals telling the device which pixel ought to be considered next (GETPIXEL); and the third (SENDPIXEL) for sending to the screen the coordinates of the pixel, the ID of the visible object (for being able to point to objects through the use of a light-pen), the depth of the displayed point, and its color. A formal definition of the device is given below where the device is treated as a single process called VIEW.

PROCESS VIEW.

DATA.

ENTITIES: triangles(i) = (id(i), p1(i), p2(i), p3(i), c(i)) for
i=1, 2..., N

INVARIANT: id(i) in IDSET

p1(i), p2(i), p3(i) in [-k, +k](3)

c(i) in COLORSET

PROCEDURES.

OBJTRANS

INTERFACE: OBJTRANS(objectid, transformation)

INPUT-ASSERTION: objectid in IDSET function(transformation) = T
in T where T is the set of all possible object
transformations

OUTPUT-ASSERTION: for every i:
id(i)=objectid=>triangle(i)=(id(i),t(p1(i),p2(
p3(i)),c(i))

RETURN-VALUE: none.

VISIBLE.

INTERFACE: (id, z-coord, color)=VISIBLE(current-x-coord,

current-y-coord)

INPUT-ASSERTION: $1 \leq \text{current-x-coord} \leq m$

$1 \leq \text{current-y-coord} \leq n$

OUTPUT-ASSERTION: true

RETURN-VALUE: $\text{id} = \text{id}(i)$

$\text{color} = c(i)$

$z = \text{intersect}(\text{triangle}(i), \text{current-x-coord}, \text{current-y-coord})$

where $\exists(j): z \leq \text{intersect}(\text{triangle}(j), \text{current-x-coord}, \text{current-y-coord})$

GETCMND.

INTERFACE: GETCMND ? (objectid, transformation)

INPUT-ASSERTION: true

OUTPUT-ASSERTION: true

RETURN-VALUE: objectid ? IDSET

$\text{function}(\text{transformation}) = t ? T$

where T is the set of all possible object transformations and $t: [-k, +k](3) ? [-k, +k](3)$

GETPIXEL.

INTERFACE: GETPIXEL ? (current-x-coord, current-y-coord)

INPUT-ASSERTION: true

OUTPUT-ASSERTION: true

RETURN-VALUE: $1 \leq \text{current-x-coord} \leq m$

$1 \leq \text{current-y-coord} \leq n$

SENDPIXEL.

INTERFACE: SENDPIXEL ! (id, x-coord, y-coord, z-coord,
color)

INPUT-ASSERTION: id ? IDSET

$1 \leq \text{x-coord} \leq m$

$1 \leq \text{y-coord} \leq n$

$-K \leq \text{z-coord} \leq +k$

color ? COLORSET

OUTPUT-ASSERTION: true

RETURN-VALUE: none.

BEHAVIOR.

(GETPIXEL ?) GETPIXEL ?(x,y)<<(id,z,c)=VISIBLE(x,y)<<
SENDPIXEL!(id,x,y,z,c

(GETCMND ?) GETCMND ?(obj,t)

(x=y=1)GETCMND?(obj,t)<<OBJTRANS(obj,t)

(x=y=1)OBJTRANS(obj,t)<<VISIBLE(x,y)

2.2.1.2 Net Definition

Several processes communicating among each other form a net. It represents an accurate model of a system whose data and processing are both distributed. Furthermore, by decomposing component processes into respective nets, a hierarchical description of the system may be constructed. This, in turn, allows the designer to systematically evaluate distinct levels of distribution: geographic, local, etc.

The approach described here differs from other such as, RSL [8], DREAM [7], and PLITS [14] in two fundamental ways. Firstly, it is the very strong emphasis on the separation of concerns principles as reflected by the process definition and also manifest in the definition of the net. Secondly, it is the flexibility to choose, within a net, any communication scheme deemed appropriate for representing the system at hand.

DEFINITION:

A net is defined as:

$$n = (P, G, C)$$

where

P is a finite set of processes.

G is an interconnection graph which maps each receiving port of some process from P to at most one sending port of some other process in P.

C is a communication model establishing the order relationship

between send type events and receive type events. Behavior expressions are used to define C in a manner similar to the process behavior. Events that may be used to define C are the send and receive events of processes from P and also additional events required to achieve specific communication protocols.

By using the behavior expressions, the communication protocol used by Hoare in [15] could be simply written as follows:

send.to.b! (x) < receive.from.a? (y) with y=x

A less restrictive protocol could allow the sending process to continue before the actual receipt occurs, if it ever does;

send.to.b! (x) << receive.from.a? (y) with y=x

(Note that proper ordering of receipts is not guaranteed).

It is now possible to present a distributed version of the hidden surface elimination device.

The distributed solution assumes the existence of n processes, ($1 \leq i \leq n$), one for each triangle, forming a unidirectional pipeline. Messages flowing along the pipe (from i to $i+1$) contain either commands or the coordinates of the current pixel $P(xy)$ along with the depth, color, and object name associated with that pixel, so far. For each pixel $P(xy)$, a process i checks the line segment starting from the viewpoint and passing through the point (x,y) on the screen for intersection against the triangle it controls. Whenever one or more intersections are detected, the depth and color of the intersection closest to the viewer are computed and compared with the data received from the left neighbor. The data sent to the right neighbor is adjusted accordingly. While i computes the depth and color of the i 'th triangle

relative to pixel $P(xy)$, its left neighbor, $i-1$, is already considering the $(i-1)$ 'th triangle against the pixel following $P(xy)$. Thus, at the end of the pipe, each pixel's minimum depth, corresponding color, and object affiliation arrive at regular intervals and are placed in the buffer of some display processor.

Example (Part 2)

NET VIEW.

PROCESSES.

PROCESS UNIT_i.

DATA.

ENTITIES: triangle = (id, p1, p2, p3, C)

INVARIANT: id ? IDSET

p1, p2, p3 ? [-k, +k](3)

c ? COLORSET

PROCEDURES.

TRANS.

INTERFACE: TRANS (transformation)

INPUT-ASSERTION: function (transformation) = t t?T where T is
the set of all possible transformations, and t:
[-k, +k](3) ? [-k, +k](3)

OUTPUT-ASSERTION: triangle = (id, t(p1, p2, p3), c)

RETURN-VALUE: none

DEPTH.

INTERFACE: $z = \text{DEPTH}(\text{current-x}, \text{current-y})$

INPUT-ASSERTION: $1 \leq \text{current-x} \leq m$

$1 \leq \text{current-y} \leq n$

OUTPUT-ASSERTION: true

RETURN-VALUE: $z = \text{intersect}(\text{triangle}, \text{current-x}, \text{current-y})$

GETCMND.

INTERFACE: GETCMND? (objectid, transformation)

INPUT-ASSERTION: true

OUTPUT-ASSERTION: true

RETURN-VALUE: objectid ? IDSET

function (transformation) = t

t?T where T is same as above

PASCMND.

INTERFACE: PASSCMND! (objectid, transformation)

INPUT-ASSERTION: objectid ? IDSET

function (transformation) = t

t?T where T is same as above

OUTPUT-ASSERTION: true

RETURN-VALUE: none

GETPIXEL.

INTERFACE: GETPIXEL? (current-x-coord, current-y-coord)

INPUT-ASSERTION: true

OUTPUT-ASSERTION: true

RETURN-VALUE: $1 \leq \text{current-x-coord} \leq m$

$1 \leq \text{current-y-coord} \leq n$

SENDPIXEL.

INTERFACE: SENDPIXEL! (id, x-coord, y-coord, z-coord, color)

INPUT-ASSERTION: id ? IDSET

$1 \leq \text{x-coord} \leq m$

$1 \leq \text{y-coord} \leq n$

$-k \leq \text{z-coord} \leq +k$

color ? COLORSET

OUTPUT-ASSERTION: true

RETURN-VALUE: none

COMPARE.

INTERFACE: (new.id, new.z, new.c) = COMPARE (c.id, c.z,
c.c, z)

INPUT-ASSERTION: c.id ? IDSET

$-k \leq c.z \leq +k$

$-k \leq z \leq +k$

c.c ? COLORSET

OUTPUT-ASSERTION: true

RETURN-VALUE: new.id = c.id

new.c = c.c if $z \leq c.z$

new.z = c.z

new.id = id

new.c = c if $z < c.z$

new.z = z

BEHAVIOR.

(GETCMND?) GETCMND(oid,t)

(oid=id & c.z=c.y=1) GETCMND(oid,t)<<TRIANGLE)

(oid=id & c.x=c.y=1)TRANS(t)<< z=DEPTH(c.x,c.y)

GETCMND(oid,t)<< PASSCMND(oid,t)

(GETPIXEL?) GETPIXEL?(c.id, c.x, c.y, c.z, c.c)

<< z=DEPTH(c.x, c.y)

<< (new.id, new.z, new.c) = COMPARE(c.id,
c.z, c.c, z)

<< SENDPIXEL(new.id, c.x, c.y, new.z, new.c)

LINKS

GETCMND of i to PASSCMND of i-1 unless i=1

GETPIXEL of i to SENDPIXEL of i-1 unless i=1

COMMUNICATION

PASSCMND(i) (oid(i), t(i)) << GETCMND(i+1) (oid(i+1), t(i+1)) with
oid(i) = oid(i+1) t(i)=t(i+1)

SENDPIXEL(i) (id(i), x(i), y(i), z(i), c(i) << GETPIXEL(i+1)
(id(i+1), x(i+1), y(i+1), z(i+1), c(i+1)) with id(i) = id(i+1)
x(i) = x(i+1) y(i) = y(i+1) z(i) = z(i+1) c(i) = c(i+1)

2.2.2 Performance Specifications

Performance is defined here as the ability of some system to meet a set of stated constraints. The constraints are assumed to be expressed in objective, precise terminology even when their origin might be entirely subjective.

The nature of the stated constraints is the determining factor for the manner in which the performance checks need to be carried out. One taxonomy of constraints may divide them into quantitative and qualitative. The quantitative constraints place bounds on various performance indices such as those used to measure productivity, responsiveness, or utilization. The indices are derived by employing mathematical and/or simulation techniques in the context of particular system models. The qualitative constraints, on the other hand, specify non-numeric properties of the system, properties that can be checked by analyzing the system or its specifications. Examples of qualitative constraints are single fault recovery, hardware regularity, and single source procurement.

Another way of categorizing constraints is by considering their relevance domain rather than their nature. The new taxonomy could, for instance, separate performance constraints into:

- (1) Constraints pertinent to the technical aspects of the system.
- (2) Constraints regarding the system's development (they bring into play factors such as time, cost and resources, aspects which undoubtedly have a significant impact on the design decision making process).
- (3) Constraints related to product enhancement and maintenance.

- (4) Constraints over the potential environmental impact such as personnel retraining, human interfacing, etc.

Such an explicit and broad constraints-oriented perspective on performance is not too common in current literature. As a matter of fact, work pertinent to various classes of constraints fall into separate (often isolated) research areas (e.g., fault tolerance, performance modeling and simulation, reliability, software economics, etc.). Growing emphasis on building formal systems specifications, however, is certain to produce a unified view of performance through the development of functional/performance specifications.

While the advantages of basing performance evaluation directly on the functional specification have been recognized for quite awhile, there are few published results. Automatic generation of simulation models have been described in conjunction with some specification languages such as RSL [8] and DREAM [7]. More recently, the concept of performance abstract data type has been proposed by Booth and Wiecek [13].

Fundamental to the development of performance specifications is the question of how they ought to be combined with the functional specifications. This is the main issue addressed by this section which advocates the notion of extending the functional specifications to include performance attributes and constraints. The latter are to be stated as predicates over the functional characteristics of the system and the relevant performance attributes.

DEFINITION:

A process whose specification includes performance data is called a performance process. A formal definition is given below.

$\tau p = (p, W, E, K)$

where

- p is a process
- W is a set of attributes some of which are basic while others are derivable from the ones that are basic.
- E is a performance evaluation model which describes the manner in which basic attributes are related to entities of the process and the rules needed to compute the derived attributes.
- K is a set of constraints expressed as predicates over the values associated with the members of W.

2.2.2.1 Performance Process Definition

For the sake of clarity this section considers the way performance attributes and constraints relate only to process specification. Nevertheless, it is important to keep in mind at all times the dynamics of constraints propagation. During top-down design, constraints over some level n are checked for satisfiability by making reasonable assumptions about some of the attributes. These assumptions reflect performance expectations with respect to lower levels of the design and, thus, they transform into constraints that are imposed by level n on subsequent levels. Once the level of detail suffices as to replace some assumptions with actual performance data (i.e. validated assumptions), an upward flow of corrected performance information may be generated thus providing increased accuracy to the results of earlier analyses. If some of the

assumptions, however, are proven wrong a certain amount of redesign is needed. The emphasis on top-down propagation of constraints marks a departure from Booth and Wiecek approach that seems to focus primarily on the nature of the bottom-up flow of performance information.

The basic attributes are the key to both top-down propagation of constraints and the bottom-up flow of validated performance data. They bias design choices toward those that seem to make the safest assumptions about basic attributes and also have least difficulty in meeting the constraints, given those assumptions. Any such performance process for which the constraints are met, given the assumptions made about the basic attributes, will be called a partially satisfied process. In contrast, a totally satisfied process denotes a process whose constraints are met by a complete set of validated attributes. It is the result of the bottom-up propagation of validated assumptions.

The nature of the basic attributes is quite diverse. It depends, first of all, upon the functional entities which they characterize. Data attributes (such as storage space or maximum record size), for instance, need to characterize both assumptions about invariant properties of the data and the temporal qualities affected by procedure invocations. Attributes associated with various procedures (e.g., delay) typically depend upon the input/output parameters and the data entities involved, while behavior attributes (e.g., average response time) need to relate to the procedures being sequenced.

The nature and objectives of the evaluation are another factor. Different attributes are considered when constraints

are over throughout rather than development time or reliability. Furthermore, the use of a deterministic versus stochastic approach also alters the type of the basic attributes, as illustrated by the two examples below.

Example (A performance process employing a deterministic performance evaluation model).

PROCESS SAMPLE1

DATA.

ENTITIES: a ATTRIBUTE size

INVARIANT: ...

PROCEDURES.

GET.

INTERFACE: GET?(x) ATTRIBUTE time(GET) = 5*size(x)

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...X... ATTRIBUTE size

UPDATE.

INTERFACE: y = UPDATE(v) ATTRIBUTE time(UPDATE) =
size(a)/2 + size(v)

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...y... ATTRIBUTE size(y) = 2

CORRECT.

INTERFACE: z = CORRECT(w) ATTRIBUTE time(CORRECT)
 = 3*size(w)

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...z... ATTRIBUTE size(z) = 1

SEND.

INTERFACE: SEND!(u) ATTRIBUTE time(SEND) = 3

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...

BEHAVIOR.

(GET?) GET?(x) < y = UPDATE(x)

(y = UPDATE(x) << z = CORRECT(y)) v (y = UPDATE(x) << z =
CORRECT(x))

y = UPDATE(x) << SEND!(y) << GET?(#)

z = CORRECT(#) << GET ?(#)

CONSTRAINTS. (status = totally satisfied)

"Average response time, *artime*, must be shorter than 800".

$artime < 800$

where *artime* is defined as the average time between consecutive invocations of the GET procedure when data is assumed to be always available for being read in.

ASSUMPTIONS.

$size(a) = 1000$ (basic-validated)

$size(x) = 10$ (basic-validated)

$artime \leq 593$ (derived-validated)

"The validation of *artime* is based on the following

$artime \leq time (GET?(x)) + time (y = UPDATE(x)) +$

$max (time (z = CORRECT(y)), time (z = CORRECT(x))) +$

$time (SEND!(z))$

$= 5 * size(x) + size(a)/2 + size(x) +$

$3 * max (size(y), size(z)) +$

3

$= 593$

This formula for artime is deductible from the definition of the process behavior".

Example (A performance processing employing a stochastic performance evaluation model.)

PROCESS SAMPLE2

DATA.

ENTITIES: a ATTRIBUTE size

INVARIANT: ...

PROCEDURES.

GET.

INTERFACE: GET?(x) ATTRIBUTE time(GET) =
 5*size(x)

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...x... ATTRIBUTE size

UPDATE.

INTERFACE: y = UPDATE(v) ATTRIBUTE time(UPDATE) =
 size(a)/2 + size(v)

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...y... ATTRIBUTE size(y) = 2

CORRECT.

INTERFACE: z = CORRECT(w) ATTRIBUTE time(CORRECT)
 = 3*size(w)

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...z... ATTRIBUTE size(z) = 1

SEND.

INTERFACE: SEND!(u) ATTRIBUTE time(SEND) = 3

INPUT-ASSERTION: ...

OUTPUT-ASSERTION: ...

RETURN-VALUE: ...

BEHAVIOR.

(GET?) GET?(x) < y = UPDATE(x)

(y = UPDATE(x) << z = CORRECT(y) ATTRIBUTE prob1) or (y =
UPDATE(x) << z = CORRECT ATTRIBUTE prob2 = 1 - prob1)

y = UPDATE(x) << SEND!(y) << GET ?(*)

$z = \text{CORRECT}(*) \ll \text{GET } ?(*)$

CONSTRAINTS. (status = partially satisfied)

"Average response time, artime , must be shorter than 580".

$\text{artime} < 580$

where artime is defined as the average time between consecutive invocations of the GET procedure when data is assumed to be always available for being read in.

ASSUMPTIONS.

$\text{size}(a) = 1000$ (basic-validated)

$\text{size}(x) = 10$ (basic-validated)

$\text{prob1} = 14/24$ (basic-assumed)

$\text{prob2} = 10/24$ (derived)

$\text{artime} = 579$ (derived)

"The derivation of artime is based on the following

$\text{artime} = \text{time}(\text{GET?}(x)) + \text{time}(y = \text{UPDATE}(x)) +$

$\text{prob1} * \text{time}(z = \text{CORRECT}(y)) +$

$\text{prob2} * \text{time}(z = \text{CORRECT}(x)) +$

$\text{time}(\text{SEND!}(z))$

$$\begin{aligned}
&= 5 * \text{size}(x) + \text{size}(a)/2 + \text{size}(x) + \\
&\text{prob1} * 3 * \text{size}(y) + \\
&(1 - \text{prob1}) * 3 * \text{size}(x) + \\
&3 \\
&= 50 + 500 + 10 + \text{prob1} * 6 + (1 - \text{prob1}) * 30 + 3 \\
&= 593 - 24 * \text{prob1}
\end{aligned}$$

which results on

$$\text{prob1} \geq 13/24$$

Assuming $\text{prob1} = 14/24$, $\text{artime} = 579 < 580$ ".

The two examples above demonstrate the feasibility and some of the advantages derived from the use of functional/performance specification. There are however major issues that still await resolution. Among them, an important one is the (partial and total) satisfiability of the constraints. The examples employed an analytic approach but, depending upon the nature of the problem or the methodology being used, simulation may be required. The simulation itself may need to be (1) trace-driven when real data is available, (2) program-driven when the workload is describable in some problem-oriented language or (3) distribution-driven when stochastic models are used to represent certain input attributes. A facility meant to automate the satisfiability checks most certainly would need to incorporate the entire range of tools.

The same specifications and tools should be expected to support not

only the design proper but also the performance of hardware/software trade-offs. The case study will show how machine models could be employed to determine the feasibility of placing one or more processes on the same physical processor. In general the models may vary considerably in nature from an n-tuple of machine attributes such as (instruction execution time, core size, disk access time, disk storage size) to a detailed specification in a hardware specification language such as SMITE [18]. In the former case, the matching of the machine to the performance process can be verified analytically, while in the latter case critical software components are actually executed by emulating the described hardware and monitoring the performance.

2.2.3 References

- [1] Ross, D. T. and Schoman, K. E., "Structured Analysis for Requirements Definition", IEEE Trans. on Soft. Eng. SE-3, No. 1, pp. 6-15, January 1977.
- [2] Robinson, L. and Levitt, K. N., "Proof Techniques Hierarchically Structured Programs", CACM. 20, No. 1 pp. 271-283, (April 1977).
- [3] Liskov, B. H. and Berzins, V., "An Appraisal of Program Specifications", Research Directions in Software Technology, P. Wegner (Editor), pp. 276-301, MIT Press, 1979.
- [4] Liskov, B. H., Snyder, A., Atkinson, R., and Schaffert, C., "Abstraction Mechanisms in CLU", CACM 20, No. 8, pp 564-576, August 1977.
- [5] Wulf, W. A., London, R. L., and Shaw, M., "An Introduction to the Construction and Verification of Alphard Programs", IEEE Trans. on Soft. Eng., SE-2, No. 1 pp. 253-265, (December 1976).
- [6] Campbell, R. H., and Kolstad, R. B., "Path Expressions in Pascal", Proc. 4th Tulp. Conf. on Soft. Eng. pp. 212-219, 1979.
- [7] Riddle, W. E., Wiledon, J. C., Sayler, J. H., Segal, A. R. and Stavely, A. M., "Behavior Modeling During Software Design." IEEE Trans. on Software Engineering. SE-4, No. 4, pp. 283-292, July 1978.

- [8] Bell, T. E., Bixler, D. C. and Dyer, M. E., "An Extendable Approach to Computer-Aided Software Requirements Engineering", IEEE Trans. on Soft. Eng., SE-3, No. 1, pp. 49-60, January 1977.
- [9] Owicki, S. S. "Axiomatic Proof Techniques for Parallel Programs". Ph.D. Thesis, Report TR-75-251, Cornell University, July 1975.
- [10] Greif, I., "A Language for Formal Problem Specification." CACM 20, No. 12, pp. 931-935, December 1977.
- [11] Wegbriet, B., "Verifying Program Performance", CACM, No. 4, pp. 691-699, (October 1976).
- [12] Guttag, Z., "Abstract Data Types and the Development of Data Structures", CACM 20, No. 6, pp. 396-404, June 1977.
- [13] Booth, T. L., and Wiecek, C. A., "Performance Abstract Data Types as a Tool in Software Performance Analysis and Design", IEEE Trans. on Soft. Eng., SE-6, No. 2, pp. 138-151, March 1980.
- [14] Feldman, J., "High Level programming for Distributed Computing", CACM 22, No. 6, pp. 353-368, June 1979.
- [15] Hoare, C. A. R., "Communicating Sequential Processes", CACM 21, No. 8, pp. 666-677, August 1978.
- [16] Shaw, A., "Software Descriptions with Flow Expressions", IEEE Trans. on Soft. Eng., SE-4, No. 3, pp. 242-254, May 1978.

- [17] Roman, G. C. and Kimura, T., "AVLSI Architecture for Real-Time Color Display of Three-Dimensional Objects", Proc. of Delaware Bay Computer Conference, March 1979.
- [18] McClean, R. K. and Press, B., "The Flexible Analysis, Simulation and Test Facility: Diagnostic Emulation", Technical Report TRW-SS-75-03. TRW Defense and Space Systems Groups, Redondo Beach, CA 90278.

2.3 H/S TRADEOFFS METHODOLOGY FUNDAMENTALS

2.3.1 Methodological Approach to Binding

The importance of the binding phase lies in the fact that it brings system development for the first time into the realm of the tangibles with which the customer is more familiar. As a consequence, however, the issues become more complex as economic, political, social, and other considerations gain significance alongside the strictly technical aspects. The significance of this phase rests, therefore, upon the impact it has on the success or failure of some system.

The complexity of binding has long been recognized [4] even for the somewhat simpler situation when a single computer system is to be purchased. The difficulties stem from the growing multitude of feasible alternatives, the virtual impossibility to objectively compare highly dissimilar systems, the lack of sophistication of the performance measurements, the complexity of evaluating cost and human factors, the subjectivity of the selection team, its role, power, and composition, etc. As one considers distributed systems and custom-made components, the complexity grows many fold with the technical expertise required of the selection team increasing accordingly.

The difficulty of binding can be alleviated to some degree when one considers as input the processing model advocated in the realization phase. The reason is to be found in the considerable amount of information regarding the characteristics of the system already available in the processing model. It includes for each processor the processes that compose it, their performance constraints and functionality, and the data involved, along with expected volume and access patterns. Furthermore, communication protocols among processors and related constraints (e.g., data rate, acceptable delays, etc.) are

also part of the model which has been subjected to extensive optimization attempts.

This section emphasizes primarily the technical issues related to binding: performance of hardware/software tradeoffs using as the objective function complexity, risk and development cost minimization. The fundamental assumption is that, based on this approach, several alternatives may be chosen in an objective manner, leaving the final decision-making body to consider fewer configurations and reevaluate them based on additional less quantifiable criteria (e.g., vendor's reputation, future needs, maintenance costs, need for standardization, delivery schedule, experience with particular hardware, economic trends, politics, etc.).

The binding strategy takes advantage of the top-down organization of the processing model in order to minimize not only the objective function but also the effort required to carry out the binding. Furthermore, various binding options are considered on a priority basis. A lower priority option is introduced only when increased levels of distribution cannot satisfy the performance needs given the current option or become prohibitive in terms of cost. Furthermore, given a feasible option, an attempt is made to minimize cost and number of components (when cost differences are small).

The number of binding (i.e., hardware/software partitioning options is surprisingly large. The more commonly encountered options are listed below:

- o All software (i.e., software on already given hardware)
- o Software/commercial-computers systems (i.e., computer system selection)

- o Software/customized-commercial minicomputer (i.e., micro-programmable minis)
- o Software/customized-firmware/custom-mini
- o Firmware/custom-hardware
- o All hardware (SSI, LSI, VLSI)

The first step of the binding phase (see Figure 2.3.1-1) identifies the minimum distribution requirements, i.e., what processors are prohibited from being implemented on the same (physical) machine. This is done by reviewing pertinent customer imposed constraints (e.g., survivability, fault tolerance, etc.) and by recognizing performance constraints which obviously require distribution. Next, tightly coupled processors of comparable structure, behavior, and performance are grouped into clusters. They represent areas of hardware regularity which ought to result in uniform binding, i.e., each processor in the cluster is bound to the same type of physical machine.

Based upon the characteristics of its members, a binding option is selected for each cluster. Every time an option is considered, selection rules for determining the set of viable candidates are established. They help in reducing the number of candidates to those that have a better than average chance of success and satisfy all the basic binding constraints. The rules have to be simple to apply, making initial candidate selection trivial.

Once the candidate set has been chosen, each candidate is evaluated with respect to the cluster. The evaluation method

depends on the current option. Its role is that of determining how good a match the candidate is for the particular cluster. Each processor is mapped into a physical machine. The mapping is verified and later evaluated with respect to performance and other qualities that the system must exhibit. Candidates that are too powerful or not powerful enough are rejected. A good match between some candidate and the cluster results in a successful binding which, along with other successful bindings, is saved for future consideration. Failure to bind at least one of the candidates results in taking one of the following three courses of action: further distribution of the processors in the cluster, selection of a different binding option, or restructuring of one or more clusters.

The strategy above results in a number of possible configurations. Not all of them are viable due to incompatibilities that may occur when communicating clusters are separately bound. As such, all incongruent configurations need to be discarded before continuing with the binding of the communication links.

At this point a final validation of each configuration takes place, and all acceptable configurations receive a cost-value coefficient. The cost estimates ought to include, in each case, hardware costs, software costs (two equally priced machines but having different software results in different software development costs), and communication costs. The final choice is determined by contrasting the cost of the various alternatives (development cost plus any other cost factors) and by considering any other issues perceived to be relevant to the decision in question. Thus, the binding approach proposed here combines in an elegant manner systematic objective evaluation and subjective selection.

The binding phase, however, cannot be completed until precise and complete software and hardware requirements are generated, thus assuring the stability of both types of components during the implementation and manufacturing phases. Unless the requirements are frozen at the end of the binding phase, the system is "bound" to fail. Hardware requirements must identify the exact hardware and accompanying software to be procured or, if custom items are present, their functionality, hardware interfaces, storage requirements, performance constraints, and configuration constraints. With respect to the software requirements (other than purchased software), they ought to state functionality, data needs, interfaces, performance constraints, and the development language. Based on these requirements, procurement and/or implementation may start without any risk.

The remaining subsections elaborate on issues specific to each of the four key binding options.

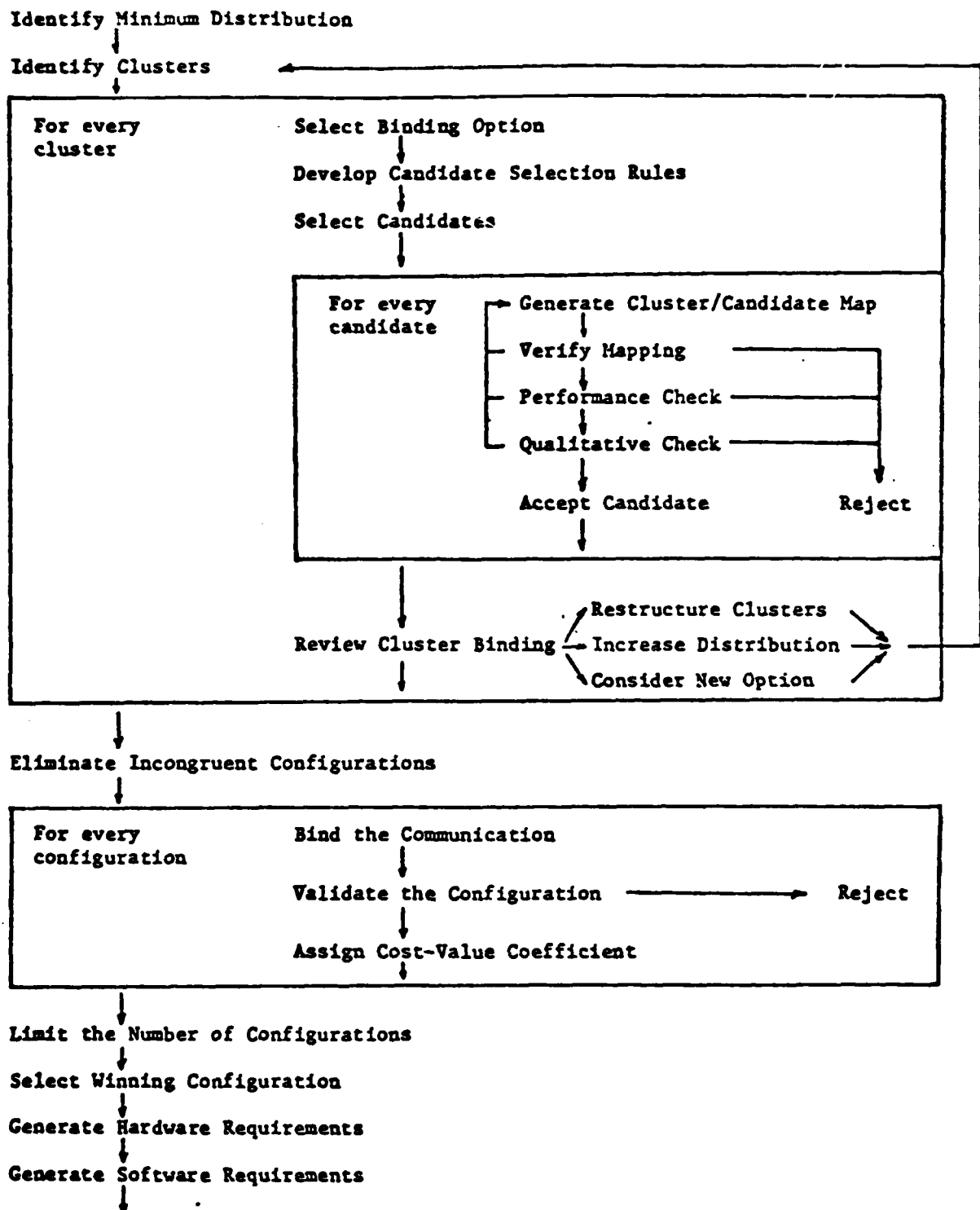


FIGURE 2.3.1-1: Binding Phase

2.3.2 Selection of Commercial Computer Systems

The selection of commercially available computer systems is the most frequent binding option encountered by system developers. Despite its practical importance and the considerable attention it has received [4, G], computer system selection continues to be carried out in an ad-hoc subjective manner. The reasons are many. Exposure to a particular vendor may breed a sense of familiarity and comfort. The objective techniques available for selection are complicated and costly. Actually, there are two distinct issues that make the selection difficult: Candidate evaluation and assignment of a cost-value coefficient.

The candidate evaluation steps are aimed at determining the ability of the candidate to support the performance and other needs of the processors involved. A survey of various approaches employed today is available in [g] along with an extensive bibliographical list. The sophistication and basic philosophy of the different approaches show tremendous variability. Some of the most primitive techniques are based on hand calculations. They require one to estimate the number of various types of computations and device accesses and to compute a total time based on the time required to perform each operation. Such measurements are suitable only for relatively simple systems.

In the category of inexpensive experimental techniques, instruction mixes and kernels are used. Small programs or sets of concurrent tasks are carefully designed so as to permit extrapolation of performance data for the case of actual large programs. A superior but also costlier group of techniques includes artificial, standard, and live benchmarks. The construction of the transportable benchmark is expensive and impractical when the software has not yet been designed. Modeling and simulation [1, 2, 6] appear to be the most promising approach and best

suited for use in connection with the data already existing in the processing model. Nevertheless, there are severe limitations in the usage of such techniques.

With respect to assigning a cost-value coefficient, objective techniques fall into two categories: the weights and score approach and the cost-value technique. The first technique starts by assigning weights to the different features or characteristics required or desired from the candidates. Subsequently, each candidate is given a score for each weighted feature or characteristic. The total (weighted) score is used to determine the finalists. The cost-value technique [4] replaces weights and scores by cost. All desirable features and characteristics are assigned a dollar value based on how much they are worth to the customer. The actual cost of the machine is reduced by the value of all desirables that come along with the machine. A few lowest cost machines become the finalists. However, the cost associated with each candidate should be based on purchase price, plus the cost of developing the software for that machine, and minus the value of desirable features. Existing software for some machines (e.g., availability of a database facility) impacts significantly the total development cost. Good software cost estimation tools are required to support such cost computations.

2.3.3 Selection of Customized and Custom-Made Machines

The option of partitioning the system's function between software and customized or custom-made machines is exercised typically in those circumstances when important special performance requirements need to be met and commercially available machines cannot satisfy them. In such cases the developer may choose to modify the firmware of some off-the-shelf available minicomputer or to specify a custom-made minicomputer. The latter solution is often adopted for airborne computers which have to meet special weight, volume, power, and reliability requirements.

Within each one of the two options, cost differences between various candidates are not very significant. As such, the key steps are the candidate selection and evaluation, i.e., finding some candidate that is able to satisfy the requirements. For customized machines, the architecture is fixed but the instruction set may be altered. Custom-made machines require one to consider both architectural and instruction set alternatives -- there is an extra degree of freedom. Nevertheless, the same candidate evaluation techniques may be employed in both cases.

The candidates may be evaluated through the use of live benchmarks. A few very sensitive software components are implemented and used as benchmarks. The ability to satisfy the performance constraints of these components assures that all other performance constraints can be met. The benchmarks may be run on simulated machines, on specially designed evaluation based testing facilities, or on prototype machines.

Effective simulation requires the availability of a good hardware description language and extensive diagnostic and monitoring facilities. Because simulations are slow and expensive, special facilities using

emulation [5, 3] have been built. They are able to increase considerably the productivity of the candidate evaluation step. Still another possibility is that of developing a prototype machine. For customized machines, all it requires is the replacement of the microcode. For custom-made machines, prototypes can be emulated or built quickly through the use of prefabricated modules such as those described in [7]. The disadvantage of the prototypes over the use of the test facilities is the lack of diagnostic and monitoring capabilities, which may result in incomplete evaluations.

2.3.4 Electing Custom-Made (VLSI) Devices

A device is perceived here as a special purpose, high performance hardware unit made out of custom VLSI chips. The ability to build such devices has increased considerably in the last few years, but this option is limited to the few that have access to chip manufacturing facilities. Furthermore, the cost and risk factors are still quite high.

The construction of such devices requires the development of parallel algorithms having topological properties amenable to effective chip layouts and exhibiting a high degree of regularity, extensive concurrency, local communication, and considerable distribution of both memory and computation. In other words, candidate selection becomes the most difficult step by evolving into a parallel algorithm development problem. The evaluation is not particularly simple, either, since it has to consider technological limitations such as minimum achievable gate delay, maximum transmission delay, maximum number of gates per chip, and maximum number of pins per chip. [8] offers the reader an example of such a special purpose VLSI device (a very high performance real-time hidden surface elimination unit for three-dimensional color graphics) including a discussion of the manner in which it was developed and the evaluation procedures employed in demonstrating the feasibility of the approach.

2.3.5 References

- [1] Browne, J. C., "A Critical Overview of Computer Performance Evaluation", Proceedings of the 2nd International Conference on Software Engineering, pp. 138-145, October 1976.
- [2] Chandy, D. M. and Yeh, R. T. (editors), Current Trends in Programming Methodology, Vol. III, Prentice-Hall, 1978.
- [3] Clark, N. B. and Troutman, M. A., "The System Architecture Evaluation Facility, an Emulation Facility at Rome Air Development Center", Proceedings of the 1979 National Computer Conference, pp. 7-12, June 1979.
- [4] Joslin, E. O., Computer Selection. Addison-Wesley, 1968.
- [5] McClean, R. K. and Press, B., "The Flexible Analysis Simulation and Test Facility: Diagnostic Emulation", Technical Report TRW-SS-75-03, Redondo Beach, California 90278, 1975.
- [6] Muntz, R. R., "Queueing Networks: A Critique of the State-of-the-Art and Directions for the Future", Comp. Surveys 10, No. 3, pp. 353-359, 1978.
- [7] Orstein, S. M., Stucki, M. J., Clark, W. A., "A Functional Description of Macromodules", SJCC 30, 1967.
- [8] Roman, G. C. and Kimura, T., "Real-Time Hidden Surface Elimination Without Sorting." Technical Report WUCS-79-2, Department of Computer Science, Washington University, St. Louis, Missouri 63130, 1979.

- [9] Timmreck, E. M., "Computer Selection Methodology", Comp. Surveys 5, No. 4, pp. 199-222, 1973.

3.0 The FAST Methodology

A formal foundation for developing a hardware/software tradeoffs methodology was established in Section 2. In this section a step by step development of the methodology is presented. The approach is begun by formally defining FAST(Flexible, Analysis, Simulation and Test). The definition is given in Section 3.1. Additional assumptions are stated, and a framework for presenting each step of the methodology is discussed. The framework gives the objective of the step (what it is suppose to do); states the input needed to develop the step, suggest procedures; tools and other techniques and the output created.

Finally, conclusion are drawn from the results of the above activity and discussed in Section 3.5.

3.1 FAST Definition

FAST is a comprehensive hardware/software tradeoffs methodology. It is comprehensive in the sense that its steps can be applied to a wide range of system development problems, (e.g, Command, Control and Communications, Weapons Embedded, General Data Processing and Distributed Data Processing systems). The methodology has been developed to cover the following system options: Commercial-off-the-shelf systems, Machines that are microprogrammable, custom-made and customized systems and system components. FAST is composed of:

- (1) A set of methodological steps for directing the tradeoffs process.
- (2) A hardware/software analysis component for selecting an overall architecture.

- (3) A set of tools and procedures for implementing the steps and performing the hardware/software tradeoffs analysis.
- (4) A set of rules for incorporating the use of the methodology into projects, and for ensuring control over and visibility into the hardware/software tradeoffs process.

Several basic assumptions are established upon which FAST is based:

- (1) The methodology is a sequence of development steps, each of which is a refinement of the previous step, and develops more detail about the system.
- (2) Each of the steps develops a formal representation of the system and gives a set of tools to operate on the representation in order to validate its developing view.
- (3) Each step is performed by following a set of rules and procedures which encompass methods for formulating the system into the proposed representation.

The interim technical report (27) detailed the motivation behind FAST. Essentially, FAST is an outgrowth of TRW's computer architecture technology studies embedded in its SMITE computer description language. The steps of FAST shown in Figure 3.1, is an augmentation of the classical software system development methodology successfully employed by TRW for several years. The methodology begins with a formal definition of system requirements, proceeds through a preliminary hierarchical functional design of the system, and reaches an iterative loop to conduct the hardware/software tradeoffs analysis. Candidates for the tradeoffs analysis in the methodology are derived from simulation studies of functional system design, and are evaluated

quantitatively using data gathered with some specified tool such as emulation techniques. The augmented development cycle interfaces with the implementation phase (Figure 3.1), using a software first "approach, to avoid the excessive time usually required for serial hardware/software development.

Essentially, the FAST Project sought to improve performance and reliability of both hardware and software, by using diagnostic tools to study different system configurations. The diagnostic emulation techniques provided the capability of having one computer "Host", behave exactly like a "target" computer. The requirements are that detailed specifications of the target system are known and that the host is microprogrammable. Software written for the target machine can be executed on the host, producing exactly the same code. Using this concept as a basis, the FAST methodology was created. By specifying an integrated set of tools (for both hardware and software development). It should be possible to serially produce hardware and software. The thrust of the diagnostic emulation, put forth by TRW, was the development of a higher order description language for programming the diagnostic emulations. The language, Software Machine Implementation Tool using Emulation (SMITE) describes the microprogram component of the diagnostic emulators (34). To realize the full potential of FAST, a need for other tools became apparent. For example, the concept of a retargetable compiler (see Figure 3.1), for automatically compiling object code. (20). The extent to which automated tools are used and at what stages they are used gives a basic foundation for making a hardware/software tradeoffs. A hardware/software tradeoffs is assumed to be made when at some point in the design process choices are made between functional system elements. Tools are used to enhance the allocation of functions and to study the behavior of them. This portion of FAST is the hardware/software tradeoffs allocation process embedded in the hardware/software analysis component. Performance specifications, management constraints, etc. make up the decision

portion of the process. To provide for system traceability and cohesiveness a set of steps are needed to show exactly when and how a process should be performed and what tools are best fitted for this purpose.

The process of integrating the above approaches to address every phase of the system development is referred to as a Total System Design methodology (TSD). The TSD has as its basis the design of systems independent of any real functional elements. Instead, as suggested in (48), the TSD is an abstraction based on a careful examination of numerous methodologies, both hardware and software, and the tools supporting them. When these methodologies and tools are integrated, the results is a Total System Design methodology framework. Extensive research in this area has been given by Roman (48), the details of which are summarized in Section 2 of this report.

The TSD supports a hardware/software tradeoffs component called Binding. Binding is that portion of the TSD where the unique characteristics of the hardware/software duality are recognized. Its major role is to assign processors to specific hardware, and then optimize the proposed system by performing a hardware/software tradeoffs analysis. Figure 3.2 outlines the steps of binding. The process begins by receiving as input a System Design Specification and its constraints. It proceeds by identifying levels of distribution within the system, performs the hardware/software tradeoffs and finally produces a set of hardware/software documents. The arrows in the diagram indicate design flexibility by allowing feedback between steps.

How exactly does FAST fit within the TSD framework? What are the differences, and what are the similarities? Is there a common thread? It will be re-emphasized here, that the TSD encompasses many methodologies and serves as a basis for linking them together in some cohesive framework. The TSD takes a global view of the design process beginning

AD-A093 680

TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CA
FAST METHODOLOGY & CASE STUDY. (U)
NOV 80

F/6 9/2

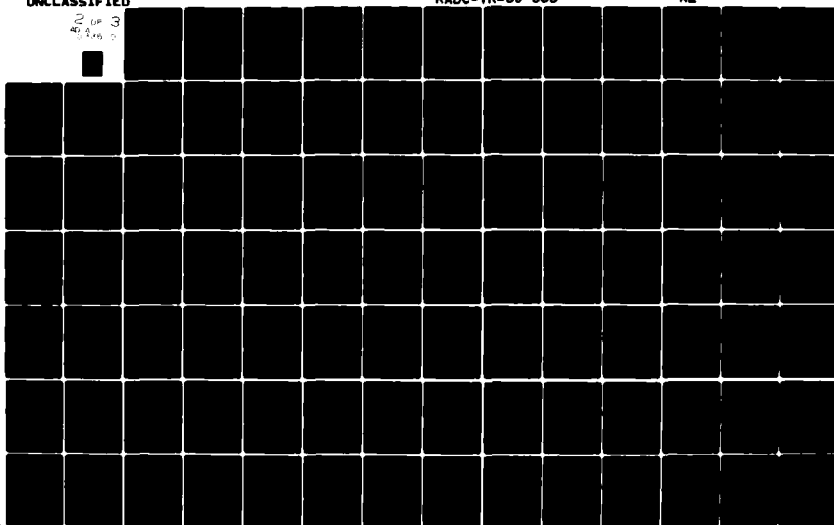
F30602-79-C-0078

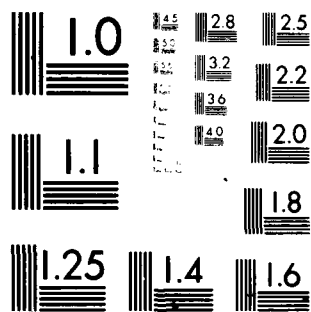
UNCLASSIFIED

RADC-TR-80-336

NL

2 OF 3
42 125 0





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

at the time a need for a system is conceived and interacts continuously with the environment throughout the life of the system. FAST is therefore a binding activity, having the same goals and system need. A careful examination of Figure 3.2, will reveal that the following corresponding steps exist between FAST and binding:

STEPS OF FAST	CORRESPONDING STEPS OF BINDING
1. Hardware/Software Partitioning	Candidate-Evaluation-Hardware/ Software Partitioning
2. Software Critical Module Benchmark for Software Development	Candidate-Evaluation-Hardware/ Software Partitioning
3. Hardware Implementation Analysis	Candidate-Evaluation-Hardware/ Software Partitioning
4. Performance Benchmark Studies	Logical Verification Performance Checker and Qualitative Checks
5. Software Concurrency Design	Generation of HIS Requirements Documents
6. Hardware Distribution and Configuration Analysis	Generation of HIS Requirements Documents
7. Distributed Preliminary Design	Section of Winning Candidates

FAST differs from binding in the limited view it takes of the partitioning activities stressed in binding. Binding provides the capability of choices between partitioning options, as well as steps for developing partitioning rules and a clearly define set of candidate selection rules. The robustness of FAST lies in the emphasis it places on the use of diagnostic emulation techniques. Binding, because it recognize various options permits flexibility in tools and techniques for performing the hardware/software analysis.

For the above reasons, we have, for the purpose of this study, extended FAST to include all of the steps of binding. The case study analysis will be performed by using the steps of Figure 3.1. The reader is referred to (27) for the developmental aspects of these steps. In this section, a summary of the steps are given in order to lay the foundation for performing the hardware/ software tradeoffs on the case study.

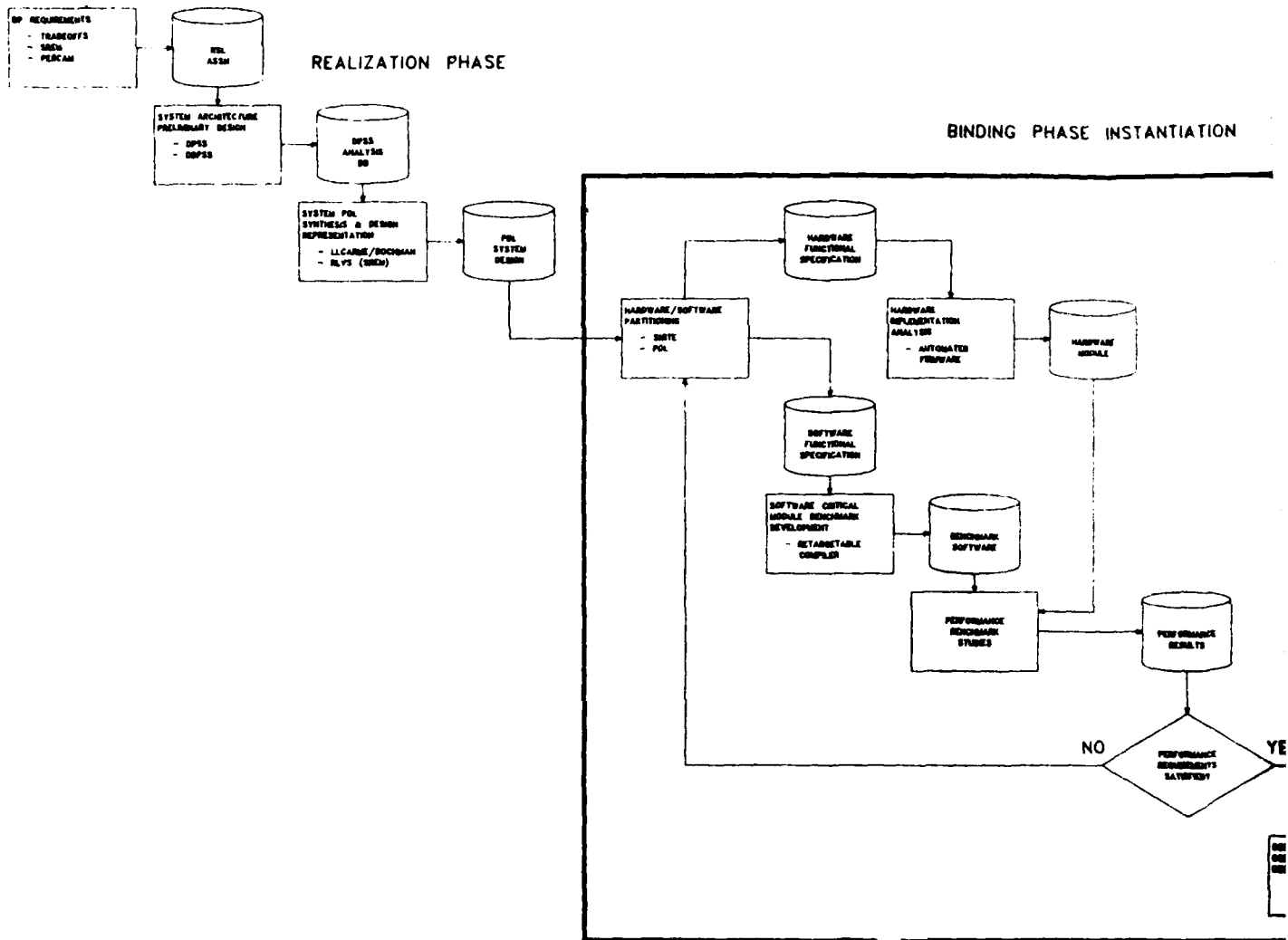


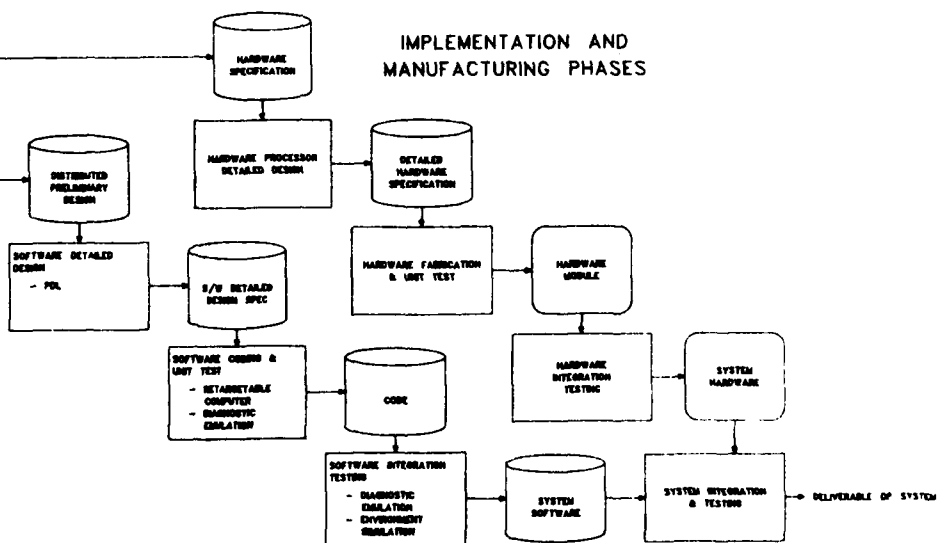
FIGURE 3.1
FAST METHODOLOGY CON

YES

SOFTWARE
EMERGENCY
MODE
- CPOL

HARDWARE DISTRIBUTION
AND CONFIGURATION
ANALYSIS
- EXTENDED STATE
- RETURN

IMPLEMENTATION AND MANUFACTURING PHASES



INCEPT

3.2 METHODOLOGICAL STEPS OF FAST

The purpose of this section is to explain the steps of FAST (Binding) in enough detail so that the hardware/software tradeoffs on the case study can be made, and add to the knowledge of system development and system engineering. No claim is made that this activity will produce a complete methodology that can automatically and immediately implemented. Rather, what we have attempted to do is:

- (1) Recognize the need for making hardware/software tradeoffs in the design of systems.
- (2) To provide a formal foundation for making a hardware/software tradeoffs.
- (3) To identify a particular set of tools to support the analysis.

Although research in system engineering (27), (48), (28), etc., agree that hardware/software tradeoffs are performed continuously at every design phase and at every aspect of engineering, it is particular to the binding phase as pointed out in Section 2.

The steps that follow are guidelines that attempt to answer some of the tradeoffs problems of system engineering.

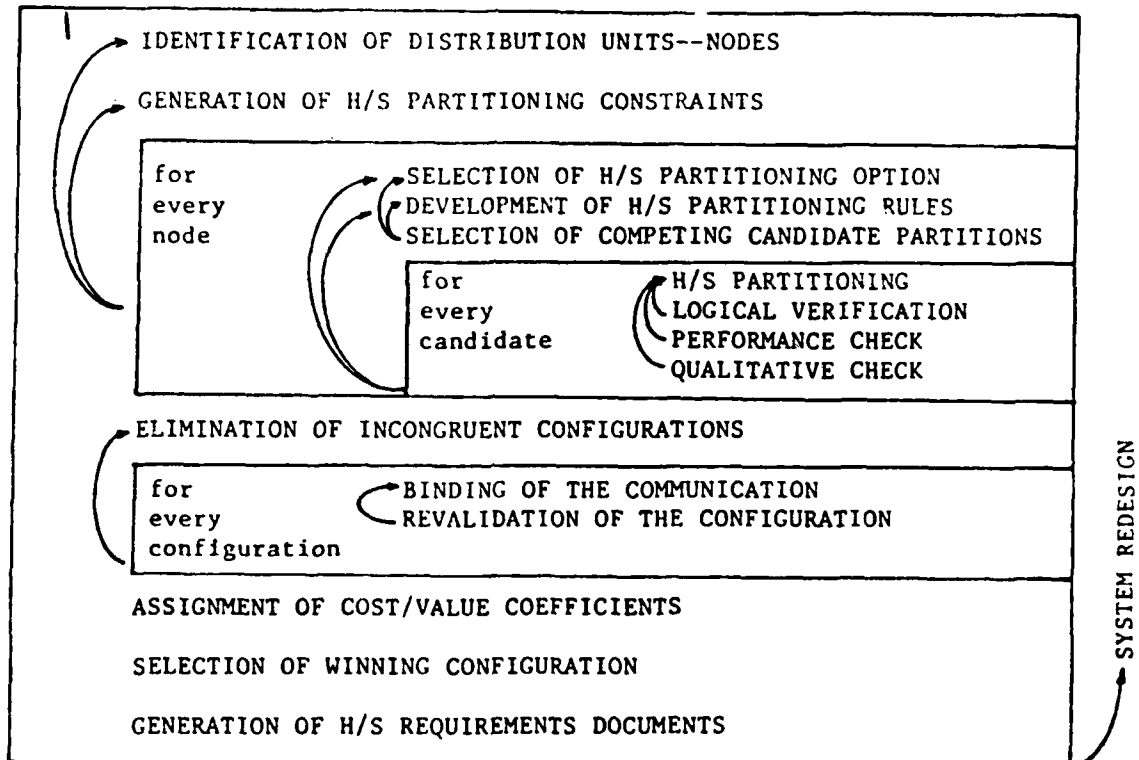
3.2.1 Identification of Distribution Units

The main purpose of the steps is to identify the levels of distribution imposed on the system. It answers the question of how many units of work is defined by the system (A node is a unit of work). What node behave similar and which one differ? How are tradeoffs made between

nodes? Classes of nodes making up the system are identified. A node may consist of a radar site, a satellite communication. It should be kept in mind that no real machine has been assigned here. The strategy is to study system behavior in terms of work to be done. Constraints plays a major role at this level. We are concerned with those constraints imposed on the system by the customer e.g., (the system must search and detect a decoy in less than two seconds). Designer constraints to meet a set of performance requirements, certain tradeoffs should be made between the nodes of work.

The importance of the step lies in its ability to link together for the first time in the development process the result of front-end development activities (the result being the system specifications) and the hardware/software tradeoffs allocation. A particular set of rules for defining how work can successfully be allocated to conceptual processors is the main tools needed to develop the step. How nodes are identified is based primarily on identification factors such as customer imposed constraints, performance constraints, etc.

INPUT: SYSTEM DESIGN SPECIFICATIONS AND CONSTRAINTS



OUTPUT: H/S REQUIREMENTS DOCUMENTS

Note: arrows represent potential backtracking.

3.2 FAST METHODOLOGY WITH ADDITIONAL STEPS

To date, no single set of tools have been developed which supports this identification activity. The activity depends on a large extent on how well the specifications were developed in the requirements phase. The output or product of this step should provide for the next set of distributions that reflect the constraints imposed upon by the system. The development of the step should provide answers to questions such as:

- (1) The number of processors to be put on what kind of hardware configurations.
- (2) What restrictions are imposed regarding the distributions?

3.2.2 Generation of Hardware/Software Partitioning Constraints

An important aspect of the tradeoffs methodology is the capability to study constraints and ascertain how they may affect partitioning options. The purpose of this step is to help identify the origin of a set of constraints and determine interdependencies between various types of tradeoffs between different nodes. The partitionings are dependent upon the system constraints and requirements. For example, if reliability is an overall constraints, then the partitioning rules should allow partitions to be grouped into a set of non-competing processes to allow modeling and other forms of evaluation techniques to be performed upon them.

Ostrowsky (38), discusses some very simple modeling procedures that may serve as potential tools to help generate the activities required by this step. Ostrowsky assigns mathematical measures to the nodes of the design. This is done by evaluating each in terms of the tradeoffs to be made, (i.e., a hardware/firmware trade that will not influence the overall system architecture), assigning a priority to each one and then treating the list mathematically. All tradeoffs in the nodes are

evaluated in this manner including the communication between nodes.

3.2.3 Selection of Hardware/Software Partitionin Options

The purpose of this step is to evaluate node, for which the partitioning rules of the above step have been applied, and make additional tradeoffs. At this point in the design process a set of well defined options are available. The most likely options are:

- All software (the software first or the hardware first).
Because of how nodes have been identified and evaluated, it may be better to design the software and select hardware based on the software.
- All hardware (this options allows tradeoffs to be studied between custom made, or commercial hardware).
- Software/firmware.
- Software/firmware/hardware.

Other options or combinations are possible.

The procedure is concerned with making sure the appropriate options are available. When a tradeoff has to be made, within a CPU, one must consider such options as: Memory space management, storage protection, etc. The options again must be evaluated in terms of the overall system mission and a lot of feedback is generated to ensure that the requirements are being meet.

This step uses a host of automated tools particularly simulation techniques where the options may be evaluated in terms of performance constraints. The possible use of such techniques are discussed in (24).

(17), and (14). There appears to be no methodology for how this is to be done and is an area that is a prime candidate for further research.

3.2.4 Development of Hardware/Software Partitioning Rules

This step becomes an important part of the methodology because of the contribution it makes to creating guidelines for selecting competing system candidates. The activities for developing the step include:

- (1) A careful evaluation of the partitioning constraints,
- (2) An examination of performance constraints and,
- (3) Those factors that are constrained by technology and the environment. The guidelines generated are the results of human judgement and what is currently available at the time of system design.

3.2.5 Selection of Competing Candidate

The hardware/software tradeoffs methodology must now provide a capability for selecting among the various candidates those would best satisfy the overall mission of the system. The evaluation process must be performed over each of the competing nodes. The step is a candidate for simulation and diagnostic emulation techniques. The step has been divided into four substeps.

3.2.5.1 Hardware/Software Partitioning

This substep involves defining all hardware, firmware and software functions based on the original requirements of the system, and under the guidance of the partitioning rules of the previous step. This is the area in the methodology where most of the hardware/software

tradeoffs techniques have evolved. For example, one may envision a diagnostic emulation facility (34), where a hardware description language such as SMITE (47), retargetable compilers and performance benchmarks techniques for evaluating the candidates are used.

3.2.5.2 Logical Verification

Added to the above set of activities is the need to allow the candidates to be treated by some logical means. This may take the form of group review, individual judgement and other management techniques to be used in system evaluations.

3.2.5.3 Performance Check

The purpose of this substep is to perform the performance evaluation model. Decisions are made as to how to emulate both the hardware performance and software performance. Decisions are optimized based on the performance evaluations.

3.2.5.4 Qualitative Check

Under this activity a further evaluation is made based on overall system requirements. Some of these qualitative factors are: Fault tolerance of the system, e.g.,

- The amount of survivability and reliability and critical functions.

Ramamoorthy and Cowan (47), have discussed a method of computing hardware and software reliability efficiency indices. The methods entail a set of equations which when computed, gives some comparison between various ways of achieving system reliability when it is known what the tradeoffs are. Time, cost, and schedules are considered under

this step.

3.2.5.5 Elimination of Incongruent Configurations

We now arrive at a point in the design process where nodes may compete with each other in the set of tradeoffs selected. Because of various technological factors one nodes, once a tradeoff is made, may be just as suitable for the job as several nodes. If the partitioning of constraints has been performed well in step 3, the task here is reduced to a minimal and eliminate conflicts between nodes. Further simulation techniques may be applied to the candidates to make final elimination and produce an optimal set of options.

3.2.5.6 Binding of the Communication and Revalidation of the Configuration

The product of the previous step is a set of system configurations. For example, we may have a distributed system, with several processors and some well defined communications between. The amount of communication is a function of system requirements, i.e., high speed communications, baudwidth), etc. The task appears to presents no substantial risk and tradeoffs when distribution is low, however, in the case of C(3) and other types of DoD systems other techniques must be considered.

Current research in this area is lacking, and there seems to be few tools specified to aid in the tradeoffs.

3.2.5.7 Assignment of Cost/Value Coefficients

One of the more detailed frameworks for developing this step is reported in the interim technical report (27), and is based on work by Joslin (50). The methodology employs a cost-value technique which

enables a user to specify desirable features of a system and make cost-tradeoffs based on the priority of these desirable features. Advantages of the techniques is its provisions for management decisions to be incorporated into the technical selection process.

3.2.5.8 Selection of Winning Candidates

The hardware/software tradeoffs process reduces to a set of computer selection techniques. Numerous methodologies are available and extensive research has been done in this area. Each company may have its own selection methodology.

3.2.5.9 Generation of Hardware/Software Documents

The final activity performed by the FAST methodology is to produce a set of hardware and software documents to be used in linking the methodology with its implementation phase.

3.3 Discussion

Presented in this part of the report has been an overview of the FAST methodology based on the formal foundation developed in Section 2. For the methodology we proposed a set of methodological steps, a hardware/software tradeoffs analysis component, and an associated set of tools and techniques. We further proposed a wide range of options that is applicable to the methodology. The methodology itself may take on several forms and its major components may vary depending upon the kind of system development under consideration. For example, in the case of a distributed systems design, the solution is complicated by the number of processors involved and the amount of interaction between them.

In order to do a meaningful analysis on the methodology, at most, another iteration through the development portion is needed. Extensive research is needed, particularly in the partitioning and generation of constraints. The hardware/software tradeoffs analysis portion of the methodology is the main candidate for using automated tools. Emulations and performance modeling techniques such as those discussed in (34) and (19), have great potential in such a methodology.

While many of the steps of FAST are applicable to a common tools approach, other may not. In the case of identification of distribution units--what tools currently exists for performing this task. How are constraints weighted in terms of the levels of distributions?

We are still left with the question of how exactly is a hardware/software tradeoffs analysis made. The answer seems to involve having available a methodology that has the facility for handling choices. A hardware/software methodology may be preceiving as an infinite set of management, technical and environmental choices. As these choices are varied over factors such as cost, performance and

schedules, the hardware/software tradeoffs methodology should be able to predict with a high degree of certainty the consequences of the choices.

Finally, we are left with the issue of where, in the system design process, does FAST fit. Our conclusion are:

- (1) FAST is an important subset of system engineering.
- (2) It links together in a precise and cohesive way, front-end development to detailed design and implementations.
- (3) It works in concert with and supports system engineering tradeoffs. It establishes methods for working with hardware and software in parallel.
- (4) If a highly effective set of automated tools can be specified, FAST can become the true portion of system engineering that can be automated.

The case study that follows will further point out how FAST may be developed as a powerful engineering tool.

4.0 A FAST CASE STUDY

This section demonstrates the feasibility and potential use of FAST by applying its methodological steps to a data processing problem. To accomplish this objective, a data processing problem for the Defense Mapping Agency's Aerospace Center (DMAAC) has been selected and the methodological steps of FAST applied to it.

The design approach used to develop the case study and perform the appropriate hardware/software tradeoffs are as follows:

- (1) A description of the DMAAC organization is given. Each of the directorates and departments making up DMAAC, and having an interest in DMIS/P (DMA Program Management Information System), is discussed. The goals, activities, input and output of each of these divisions is elaborated. The current DMSI/P system at DMAAC is also discussed pointing out the degree of automation and the responsibility of each system. (The problem environment is the output).
- (2) The second step in the case study development, is to develop an informal specification of the problem environment from the above step.
An informal design technique discussed in Section 2.2.2. The techniques employs a set of diagrams to specify each function with a directorate or division, and shows how functions interface with each other. The activity of the step results in an formal system specification of DMAAC.
- (3) This formal specification of DMAAC, provides the basis for identifying an informal specification of the DMIS/P supporting DMAAC. Again, the basis for such a specification

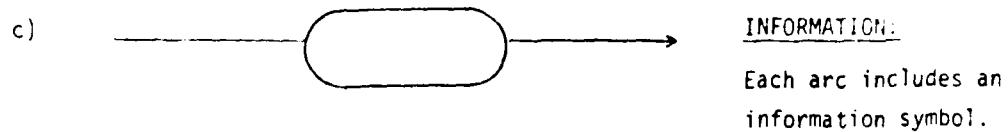
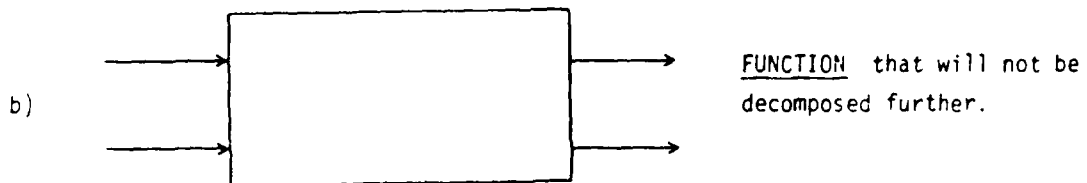
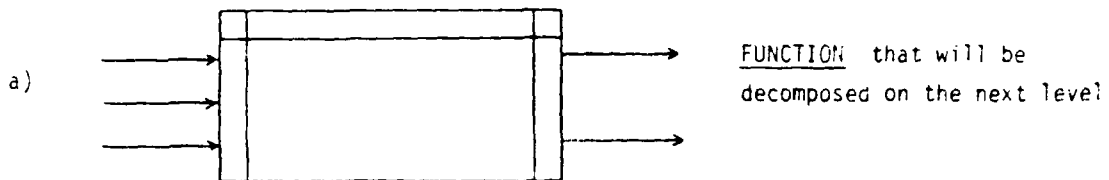
is discussed in Section 2.2.2.

A restatement of the DMIS/P is made and care is taken to ensure that all requirements by directorates and departments are included. Behavior and communication of each of the major directorates and divisions is discussed.

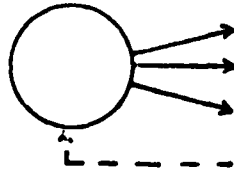
- (4) Finally, it is possible, to state the informal system in a formal way. Section 2.2.2, provides an appropriate notation for doing this. The formal specifications are then used to evaluate the steps of FAST and the hardware/software tradeoffs analysis is performed.

Formally, the DMIS/P is described in terms of data, behavior and procedures. Sections 4.1 to 4.3 details this design approach.

The following symbolism has been used to decompose the DMAAC functional flow:



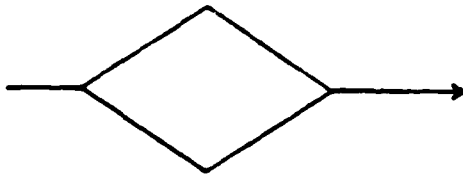
d)



EXTERNAL INPUT:

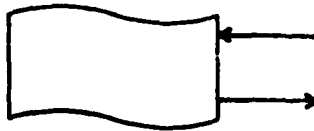
The dotted line represents
a feedback arc.

e)



CONDITIONAL: Flow of information.

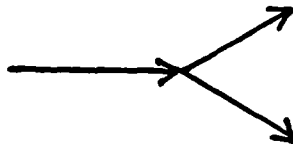
f)



PERMANENT RECORD:

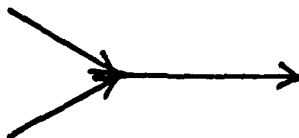
The information may be used
or updated.

g)



FORK

h)



JOIN

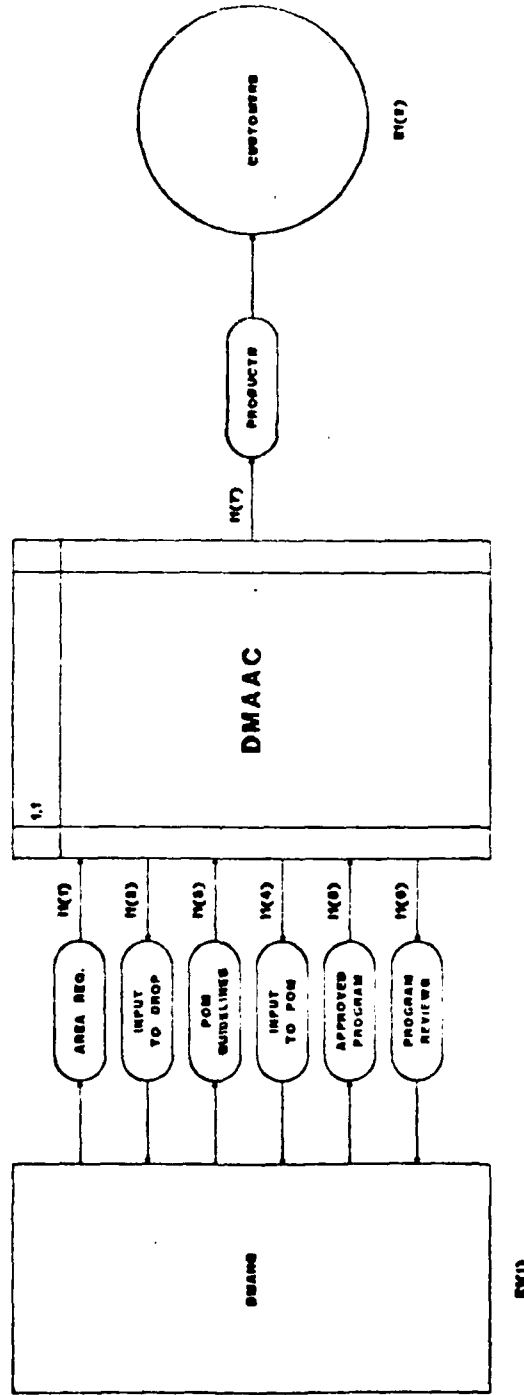
4.1 OBJECTIVE OF THE STUDY

The intent of the case study is to test the usefulness of FAST in a particular instance. Specific objectives of the case study activities are to point out the following factors:

- a. Ensure that the methodological steps are clear and consistent.
- b. Ensure that the steps and techniques are flexible.
- c. Provide for controls in the development process.
- d. Recognize and provide clear standards for documentation.
- e. Provide checkpoints and feedback allowing for frequent management reviews.
- f. Provide a consistent approach for new or additional system development, modification, and improvements.
- g. Provide the means for integrating tools and techniques as they are needed.
- h. Allow for logical decision-making.
- i. Provide economic evaluation (cost/benefit analysis).

These factors will be used as criteria for testing the FAST Methodology. If the activities of the Case Study supports this criteria it can be assumed that the methodology is feasible.

1.0 DMAAC FUNCTION WITHIN DMA ORGANIZATION



4.1 INPUT TO DMAAC

4.2 DMAAC ORGANIZATIONAL MODEL

The case selected for the study entails an examination of the data processing requirements of the Defense Mapping Agency's Aerospace Center (DMAAC). DMAAC is concerned with providing information on all charts, maps, geodesy products, production programs, status, and standards needed by the various DoD agencies. These activities are currently managed and supported by a DMA Program Management Information System (DMIS/P).

DMIS/P is one of five subsystem of DMA, and serve as a repository for information on maps, charts, and geodesy products, production programs, and related products. Those elements having an interest in DMIS/P at DMAAC are:

- a. Directorate of Programs, Production, and Operation. This directorate is responsible for preparation of the DMAAC production program. It formulates the DMAAC input to the Program Operation Management, and manages the MC&G production program, resources, and related activities. These include: allocating all center production resources required to accomplish the assigned program; provide staff coordination and supervision of the program, production and operation activities involving implementing, scheduling, and controlling work assignment; developing and implementing quality control systems; supervision of commercial contract production programs; developing and validating production and process standards; etc. The Directorate consist of three divisions:

1. Program Integration and Aeronautical Data Division
formulates and maintains the DMAAC multiyear POM and

presents it to DMAHQ; serves as a focal point for all MC&G programming, reprogramming, and production management; and assigns and manages production programs and resources.

2. Aerospace Charting Division develops, assigns, and manages multiyear production programs and resources related to navigation and planning charts; and determines objectives and priorities, application of production resources and equipment, and preparation of production specification.
3. Geopositional and Digital Data Division develops, assigns, and manages multiyear production programs and resources related to strategic and tactical point positions, point positioning data bases, digital data for simulators, and weapon system operations; geodetic/geophysical studies and applications supporting advanced weapon systems, and special projects including definition of production objectives and priorities, application of production specifications; develop and defend source data acquisition requirements essential to satisfying production program objectives.

Within PP&O, program managers are assigned responsibilities for one or more product lines. Program managers manage their program through both the program development phase and the production and program control phase. This structure is different from that at DMATC and DMAHC.

- b. Directorate of Plans, Requirements, and Technology. Has the responsibility for developing, managing, and administering

the DMA Aerospace Center mapping, charting, and geodesy (MC&G) objectives, plans, and policies with regard to operations, contingency and disaster preparedness; requirements and technology to assure readiness and availability of products and services in support of mission operations; serves as the DMA Aerospace Center focal point for MC&G intra-agency and interagency activities.

- c. Comptroller. Has the responsibility for handling financial and accounting activities, preparation of the budget, providing computer support to all nonscientific applications, and manpower, organization, and management analysis function at DMAAC.

All five production line departments were identified as having an interest in the capabilities of DMIS/P. These department include:

- a. Aeronautical Information Department is responsible for supervision and production of the FLIP products, analysis and reduction of aeronautical source data, production of aeronautical charts and aerospace charts, maintenance of the DoD Library of Free World Air Facilities and Flight Information, and providing DoD and other agencies with evaluated operational flight data and air facility information.
- b. Geopositional Department has the responsibility for providing support to manned and unmanned weapon systems by supplying geodetic, gravimetric, and geophysical data studies and technical capabilities; operations and maintenance of the DoD Library of Gravity data, generating photogrammetric control data base, producing precise point positioning data, conducting special advanced studies and satellite

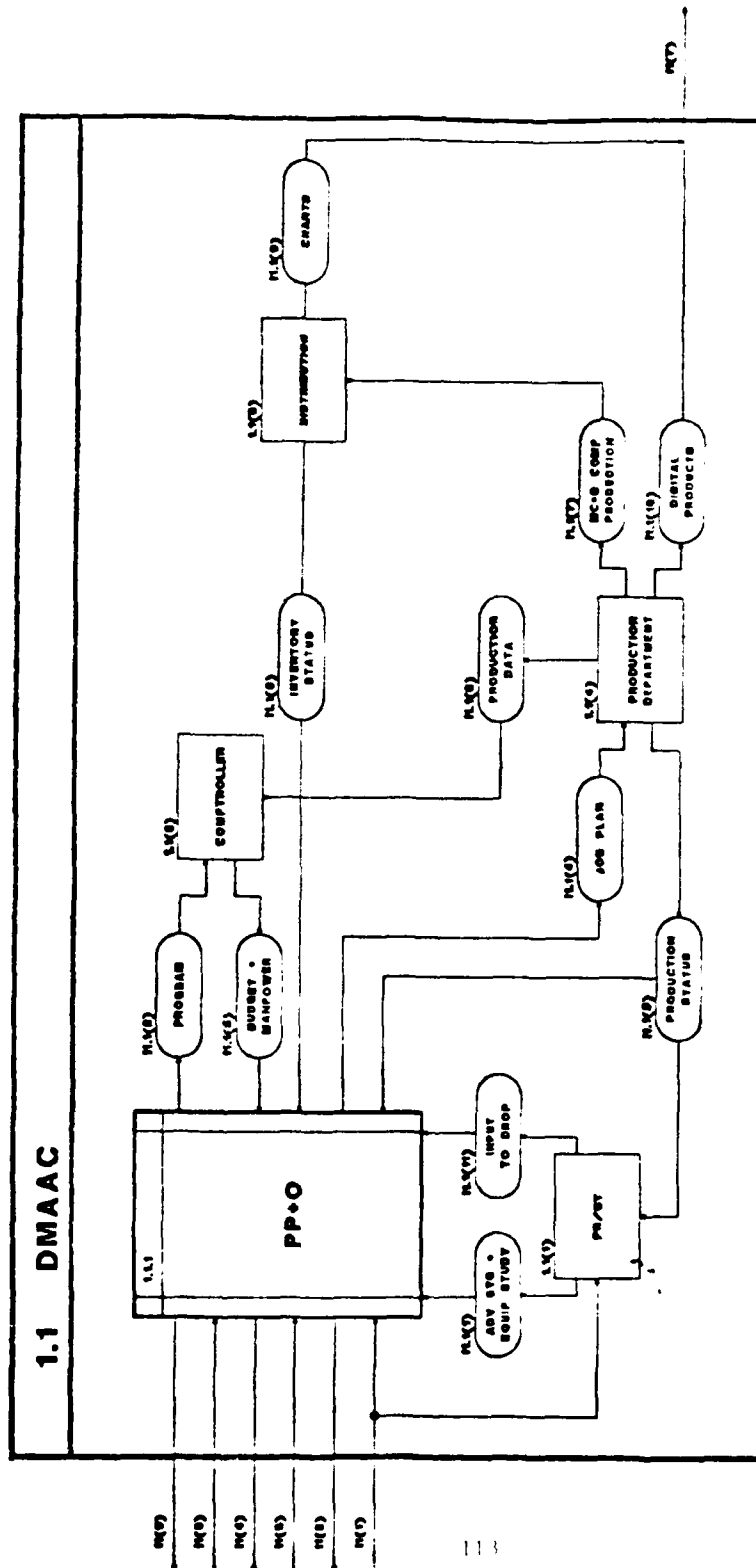
ephemorides.

- c. Aerospace Cartography Department is responsible for exploiting a variety of materials to provide stereogrammetric reductions and sensor significant imagery analysis, and planimetric and terrain data in graphic and digital format to support advanced aerospace navigation systems and simulation applications; compiling and color separating for lithographic printing of all aeronautical/aerospace charting programs, air target materials, navigation and planning charts, special purpose charts and related items by manual and automated processes; revising and maintaining published charts and graphics as required; provide technical review and guidance on cartographic work projects contracted to commercial contracts when required to supplement Department's capability, provide typographic service to center and component organizations.
- d. Scientific Data Department has the responsibility for providing support to the other line functions in the form of scientific and technical support, analysis, reduction, and evaluation of source material for cartographic products, maintenance of cartographic data base library, provision of precision photographic materials in support of production processes, and provide graphic design support of the center.
- e. Printing and Distribution Department has the responsibility for providing photolithographic products for DMA Aerospace Center; reproducing, storing, and distributing aeronautical charts, maps and publications for DMAAC primary and support missions; maintaining data required to initiate chart reprinting; preparing reports for Joint Committee on Printing (JCO) equipment.

Each production department has its own production management office which analyzes the assigned department production programs and manpower and develops production plans for the divisions. It establishes department production rates and schedules in accordance with plans and monitors all progress, and establishes department priorities. The department management officers work closely with PP&O in programming and controlling production.

The basic DMIS/P requirement by DMAAC is to have:

- o Timely output of information.
- o To provide a DMIS/P that is user oriented.
- o To provide support to levels of management requiring information.



4.2 PP&O ORGANIZATIONAL DECOMPOSITION

4.2.1 Current DMIS/P System at DMAAC

DMAAC has several automated information systems which provide program and production information to management. These include:

- a. ARAPS. ARAPS is a DMAHQ-sponsored and maintained system. Its capabilities were discussed earlier, so that they will not be discussed in detail here. The validated area requirements provided to DMAAC by DMAHQ are loaded into ARAPS by DMAAC PP&O. PP&O maintains the currency of the file by supplying updates on a monthly basis. Product status update information is supplied by ISR to ARAP via a computer tape which is generated on a monthly basis.
- b. Integrated Status Reporting System (ISRS). Both ISRS and CAPS systems are referred to as the DMIS/P systems at DMAAC. The objectives established for the ISRS system are:
 1. Provide DMAAC management with current and accurate data concerning status of jobs in progress, manpower resource requirements/utilization, manhours/dollars expended on DMAAC quality program, job completions, and progress of actual production as compared to planned production. This data is supplied to various management levels through the timely publication of a variety of reports.
 2. Eliminate or reduce where possible manual records concerning production status being maintained by various levels of management.
 3. Establish and maintain a common data base to support the current and future phases of the ISR and other major

components of Program Management Information System
(DMIS/P).

The ISRS system is designed to collect, maintain, and report information on status of jobs and resource expenditures, both planned and actual, to those organizations involved in production, planning, scheduling, and accounting. All organizations of DMAAC are involved in the system. The organizational source of the input depends upon the data being submitted. The system input can be classified into two broad categories which include:

1. Resource Utilization Data establishes and update the planned program, e.g., the types and numbers of jobs to be worked, the production schedule by month, and the type and amount of resources allocated to each. This information is compared with the actual progress of total production, obtained from the job status section of the ISR Data Base, to produce reports on status of actual production versus planned production. The Program Integration and Aeronautical Data Division (PPI) is responsible for this program data, which is generated through the CAPS Programming System.
2. Job Data Input authorizes the specific job numbers; establishes the production operations, schedules and manpower requirements for specific jobs; reports manhour expenditures, job movements, phase and job completion, and job cancellations and suspensions; and adjusts information previously entered into the ISRS Data Base. The responsibility for this category of input depends on the type of input record being generated. This responsibility ranges from the individual employee who

must report his manhour expenditures to PPI, which authorizes specific job numbers for work throughout DMAAC.

The ISRS system produces a series of output reports containing current information on assignments in the production pipeline, assignments completed, and program requirements. The output reports are grouped into three general categories: job status, job completions, and resource utilization. A general discussion of the content and usage of the three categories of output reports follow:

1. Job Status Category consist of calendar time and manhour expenditures, both planned and actual, or jobs in progress. The reports are used at various levels of management to monitor the progress of individual assignments through the various production processes, and to provide information on current and anticipated work requirements.
2. Job Completions Category has two forms. The first consists of weekly reports listing each job completed by organizational elements, along with manpower expenditures. The second form consists of a magnetic tape containing the historical summation of planned and actual expenditures of calendar time and manpower for each completed job. A current listing of the jobs contained on this history tape can be printed. This report provides historical data for standards analysis and for performance evaluation.
3. Resource Utilization Category is concerned with comparing the number of assignments in work or completed

to the number programmed for completion, and with manpower expenditures compared to allocations. This information is used to monitor the progress of actual production against programmed production.

c. Computer Assisted Programming System. The purpose of the CAP system is to:

1. Support DMAC programming and resource allocation functions performed by PPI.
2. Provide PPI with essential data for:
 - a) Preparation of the Program Objective Memorandum (POM).
 - b) Preparation of in-house production plans and general production schedules.
3. Compute the resources required to produce particular products and services; then allocate the required resources over a given time; and match the required resources with the available resources over the same time frame.

The objective of the program development phase of the programming system is to arrive at a balanced program of planned and estimated expenditures. CAPS supports this operation by supplying PP&O with nine reports which depict these expenditures and resources in different formats, summaries, and sequences.

The CAPS system consists of four subsystems which are

graphically shown in Exhibit II-17. These subsystems include:

4. The Inventory Forecast Subsystem

- a) Forecasts the resources required to complete assignments that are in various stages of the production pipeline or are assigned; calculates the resource requirements for the in-process assignments using the data in the ISR Status Master (this subsystem is not currently being used in POM development).
- b) Searches the Standards Subsystem for necessary production standards for functions that are not in ISRS because of their position in the production pipeline.

5. Program Requirements Subsystem

- a) Stores up-to-date requirements for products/services that are programmed for production during the current and subsequent six years. (Requirements expressed in number of units of product/services by quarter.)
- b) Provides:
 - Total program requirements and cumulative completions.
 - Number of items assigned for FY completions.

- Production rates based on established revision cycles and stock usage for maintenance production.

6. Standards Subsystem

- a) Contains standard data covering in-service skills, contract costs, calendar or elapsed time, and equipment to perform each work sequence for an end product or service.
- b) Contains active job plans which can be used to compare actual progress, as reported in the ISRS, to the standard processes and times contained in the Standards File for automatically generating exception reports. (This capability currently does not exist, but is planned as a long-range objective.)

7. Resource Capability Subsystem

- a) Contains the authorized manpower spaces by skill.
- b) Projects the assigned personnel within DMAAC by quarters for a two-year period and annually for the next five years. Manpower spaces and corresponding personnel status are separated into direct and indirect labor categories.

8. Programming Subsystem

- a) Combines and manipulates, in whole or in part, other CAPS subsystem content to provide production

program reports in various formats.

- b) Multiplies each new requirement from the Program Requirements Subsystem by its applicable in-service or contract standard (from the Standards Subsystem) and combines the results with the resources needed to complete the work in-process (from the Inventory Forecast Subsystem), then compares the data at various levels with the authorized and available resources (from the Resources Capability Subsystem).

The CAPS system has two options. Option 1 utilizes only the Program Requirements and Standards subsystems in completing the resource requirements. Option 2 draws on the Inventory Forecast Subsystem for calculating the resources required to complete in-process assignments, and then utilizes the Program Requirements and Standard Subsystems for calculating resources for unassigned jobs.

4.2.2 Formal Specification of DMAAC

The first step needed to accomplish the objectives of this case study is to define the problem environment. This was done in Section 4.2, where a description of the DMAAC division of DMA was discussed as well as its DMIS/P support.

The second step of the design is the development of an informal specification of DMAAC. The following steps have been taken to develop the informal specification:

- a. Define the DMAAC environment in terms of its relationship to DMA. Figure 4-1 shows this relationship.

- b. Decompose the main component of DMAAC, showing its various relationships between departments. (Figure 4.2).
- c. Finally developed a model of the DMIS/P for DMAAC organization showing. Figure 4.3 describes this relationship between directorates and departments model.

The technique used to specify the functional model is a set of top-down functional diagrams. Each diagram, whose graphix symbols are shown in Figure 4.0, results from the decomposition of a single function presented on an immediate higher level of abstraction. The "permanent record", symbol is used to indicate explicitly the memorization function while "external input" signifies an interaction with the outside environment. For example, in Figure 4.1, the function (E1) provides information I1(1) to DMAAC. Taking each of the functions within DMAAC, and using this symbolism and its decomposition strategy, the components of DMAAC have been specified by the appropriate functions and the information it sends or receives. The decomposition is shown in Figures 4.1, 4.2, and 4.3 A narrative explanation accomplishes each function.

E1(1) Defense Mapping Agency Headquarter (DMAHQ).

DMAHQ has the responsibility of developing policies, establishing program guidance, and reviewing program execution. Specifically DMAHQ has the following functions and responsibilities:

- a. Coordinate all DoD MC&G resources and activities.
- b. Provide staff advice and assistance on MC&G matters and present them to the Secretary of Defense, the Military Department, and the Joint Chief of Staffs.

- c. Develop a consolidated Mapping, Charting, and Geodesy program for review by the Joint Chief of Staff.
- d. Review and validate MC&G requirements and priorities in support of the Joint Chief of Staff for approval by the Secretary of State.
- e. Establish policies and provide DoD participation in national and international MC&G activities.
- f. Establish DoD MC&G data collection requirements.
- g. Establish DoD MC&G RDT&E requirements in coordination with the Director of Defense, Research, and Engineering and other divisions of the DoD.

The above general objectives are decomposed into a set of more specific objectives and assigned to various departments and divisions within DMAHQ and its centers, namely, DMA Aerospace Center (DMAAC), DMA Hydrographic (DMAHQ), and DMA Topographic Center (DMATC).

The interfaces (input/output) between DMAHQ is handled through the DMIS/P (Defense Mapping Agency's Program Management Information System).

The DMIS/P serves as a central repository for all information on MC&G product requirements, production programs, production capabilities, status, etc. This information is used to support resource planning, production programming and MC&G control.

I1(1) - This function represents area requirements provided by DMAHQ to DMAAC. This function initiates the program development each year by providing a set of validated DoD

area requirements. Requirements are received by Programs, Productions, and Operations Directorate (Figure 4-3), and are analyzed and reviewed. The program manager prepares the reviewed programs for input to the Resource Objectives Plan (DROP) - I1(2).

11(2) - Are the POM guidelines received by DMAAC and DMAHQ. DMAHQ provides the Program, Production, and Operation with program guidance in several forms:

- a. Information copies of the planning and guidance memorandum.
- b. Fiscal guidance by appropriation.
- c. POM Preparation instruction.
- d. MC&G Program Guidance prepared by DMAHQ, PPI.

I1(3) - Using put to DROP provided by DMAHQ in I1(3), DMAHQ develops POM guidelines and provide this input to DMAAC. DMAHQ provides PP&O with program guidance in several forms:

- a. Information copies of the Programming and Planning Guidance Memorandum.
- b. Fiscal Guidance by Appropriation to include Military pay.
- c. POM Preparation Instruction, a MC&G Program Guidance.

I1(4) - INPUT TO POM - Using guidance received from DMAHQ in I1(3), PP&O develops the POM input. Consideration is given to objectives, requirements, priorities, skill, equipment, source material availability, production schedule, training schedule, and product inventory. This function provided by the CAPS (Computer Assisted Programming System) at DMAAC.

I1(5) - This function represents input received by DMAAC from DMAHQ base on input to POM. The information consists of the approved programs, work schedules, work assignment letter, etc.

I1(6) - A final review of all programs specified in I1(5), a report is provided to DMAHQ. The specific functions within DMAAC are decomposed in and explained in 1.1 I1(1)* - indicates products provided in DMAAC and finally these products are provided to the customer.

1.1 - A representation of the functions with DMAAC. Currently DMAAC is made up of three directorates, Programs, Productions, and Operations; Program Requirements, Scientific Technology, and Comptroller, and the five production departments.

1.1.1 - Program, Production, and Operations is the focal point for all DMAAC activities. All input from and output to DMAHQ is handled by PP&O. I1(1) - I1(6), detailed in 1.0 represents this input/output relationship between DMAAC and DMAHQ.

PP&O is responsible for the preparation of DMAAC production programs. It formulates the DMAAC input to

the POM, and manages the MC&G program, resources, etc. PP&O is responsible for allocating all center production resources, provides staff coordination, production, scheduling, and controlling work assignment. Input/output is provided to the various departments written DMAAC by PP&O.

- I1.1(1)- Provides an input to Program Requirements and System Technology. This input is in the form of reports on advanced system requirements, anticipated requirements dates, as well as R&D MC&G equipments.
- I1.1(2)- A function that provides program status, productivity indexes, and manpower/skill needs for preparation of budget and manpower inventory by the comptroller.
- I1.1(3)- Represents budgets and manpower status reports prepared by the Comptroller based on input received in I1.1(2).
- I1.1(4)- This function provides input to production departments in the form of job plans. Work assignment, a list of items not meeting schedules, work assignment updates, and assignment schedules.
- I1.1(5)- Production status provides input to PP&O and PR/ST on detailed job plans, job schedules, schedule updates, and work distribution.
- I1.1(6)- An inventory status function that provides information on inventory, inventory forecast, dates, including projected out of stock dates, MC&G products, their associated stock, and the quantity needed.

- I1.1(7)- Information required by the distribution department on all MC&G products completed.
- I1.1(8)- Production data required by the production departments on production status, product status, etc.
- I1.1(9)- Charts and chart status information.
- I1.1(10)- Digital products - all information pertaining to digital products.
- I1.1(11)- Information on all PR/ST needed as input to DROP and POM.
- I1.1(1)- Directorate of Plans, Requirements, and Technology has the responsibility for developing, managing, and administering the DMA Aerospace Center Mapping, Charting and Geodesy (MC&G) objectives, plans and policies with regard to operations, contingency, and disaster preparedness, requirements, and technology to assure readiness and availability of products and services in support of mission operations; serves as the DMA Aerospace Center focal point for MC&G intra-agency activities. A major function of this department is to identify future or advanced weapon system requirements.
- 1.1(2) - Comptroller has the responsibility for handling financial and accounting activities, preparation of the budget, provides computer support to all nonscientific applications, and manpower, organization, and management analysis function at DMAAC.
- 1.1(4) - Production Departments are responsible for work assignments to specific branches and sections. Review

completions, unit produced, manpower expenditures as they are compared against standards.

Each production department has its own production management office which analyzes the assigned department production programs and manpower and develops production plans for the divisions. It establishes department production rates and schedules in accordance with plans and monitors all progress, establishes department priorities. Department management officers work closely with PP&O in programming and controlling production.

*1.1(3)- Distribution. All completed products are handled by Distribution Departments including MC&G products, digital products, and charts.

1.1.1 - Programs, Production, and Operations. PP&O interfaces with five center and departments within DMAAC. These departments are: Program Integration, Aerospace Aeronautical Division, Geopositional and Digital Data Division, A&A Product Line, and G.D.D. Product Line Department. The relationship between these departments is shown in Figure 4-3.

1.1.1(1)- Program Integration formulates and maintains the DMAAC multiyear POM and presents it to DMAHQ. The division serves as the focal point for all MC&G programming and production management; assigns and manages production programs and resources; receives input from (I1.1(2)) and provides input to (I1.1(3)) to PP&O for program, budget, and manpower information.

1.1.1(2)- Aerospace Charting Center develops, assigns, and manages

multiyear production programs and resources related to navigation and planning charts; and determines source material requirements, definition of production objectives and priorities, application of production resources and equipment, and preparation of production specifications.

1.1.1(3)- Geopositional and Digital Data Division is responsible for supervision and production of the FLIP products; analysis and reduction of aeronautical source data; production of aeronautical charts and aerospace charts; maintenance of DoD Library of Free World Air Facilities and Flight Information; and providing DoD and other agencies with evaluated operational flight data and air facility information.

The relationship between 1.1.1(1), 1.1.1(2) and 1.1.1(3) is interfaced by two product line departments; 1.1.1(4), A.A Product Line, and 1.1.1(5), G.D.D. Product Line. These product line departments plan for product plans x and y.

The above decomposition represents only those divisions with DMAAC having an interest in DMIS/P.

4.3 DMIS/P Design Specification

The design approach employed requires that the DMAAC formal specifications of Section 4.2, be stated in terms of the DMIS/P design specifications. Using the notation of Section 2.2.2, this is done by defining the directorates, (see Figures 4.1, 4.2, and 4.3) in terms of processes. Each process is primarily described in terms of the data entities it controls, its invariant properties, a set of procedures, message sending and receiving procedures, inputs and outputs and a set of rules used to describe the behavior of the process.

DMAAC requirements dictate that its DMIS/P processes communicate with each other, and; as such, form a net. (Figure 4.3). The description of the net also includes the communications between processes in the net. Processes communicate by messages. For example, PP&O may communicate with DS by sending some message (x), which is received by DS.

This input specification is created first by giving an informal description of the DMIS/P processes. This informal description is then described formally using the notation developed in Section 2.2. Such a treatment allows performance attributes to be associated with each process component, thus, establishing the foundation for making the hardware/software tradeoffs analysis.

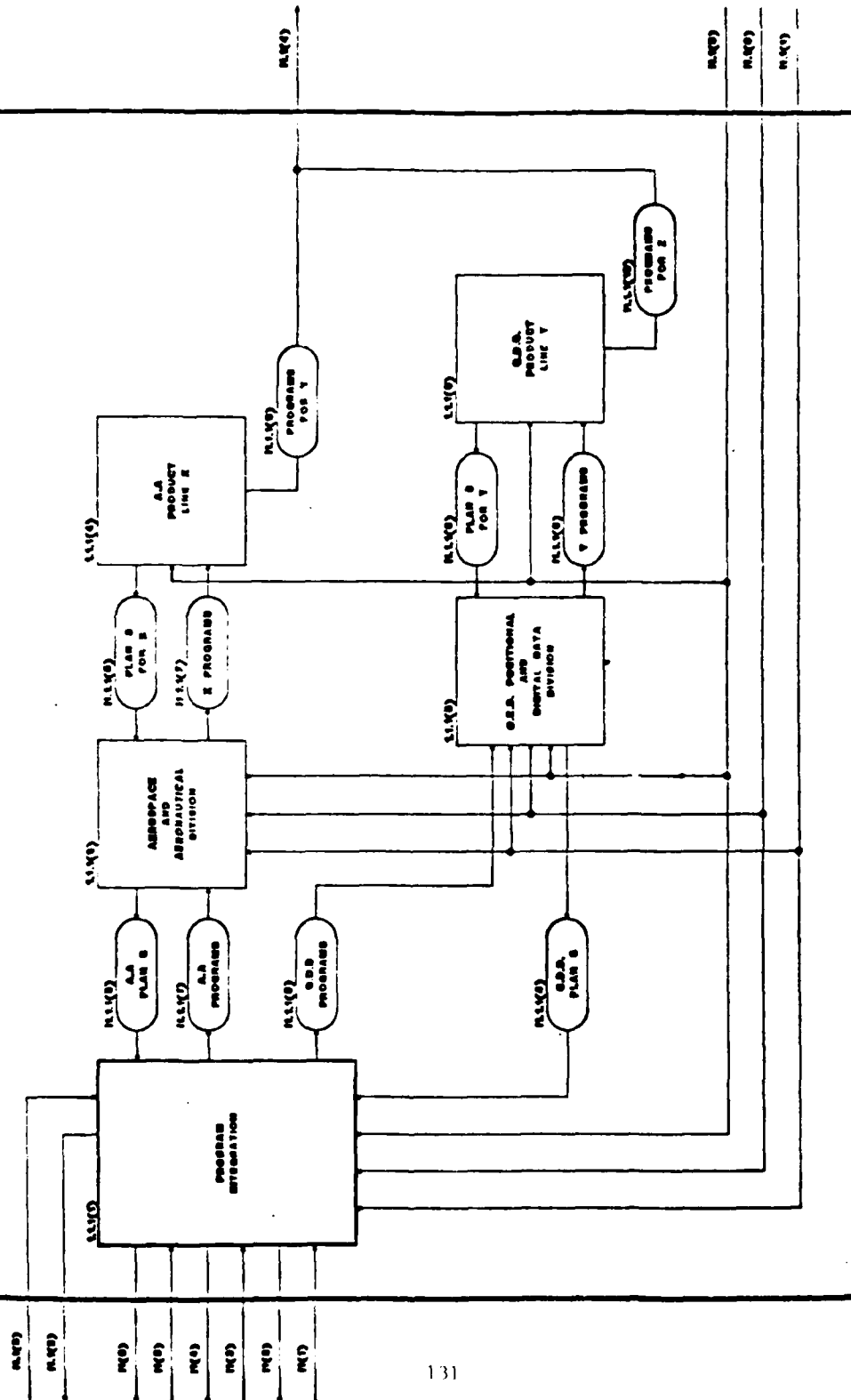
4.3.1 DMIS/P Informal Functional Specifications

Programs, Production and Operations is the most comprehensive process in the DMIS/P net. Its major function is to prepare and execute all of the DMAAC production programs. Activities supporting this function are: Program allocations, program supervision and coordination and production quality control. PP&O communicates with the other

processes in the DMIS/P net by sending and receiving messages pertaining to program and production requirements, plans, inventory and resources. Behavior within PP&O is a function of the order in which resources and data is received. For example, production requirements are received from DMAHQ, validated by PP&O, and communicated to PDS. PP&O uses its data for planning, maintaining and retrieving.

Job plans, job status, resource utilization, digital data and inventory information is handled by the various production departments with the DMIS/P net. Each production department analyzes production program and job plans received from PP&O, and in turn, sends production status to PP&O. Behavior within the various production departments is dependent on monitoring product data, processing time cards and servicing PP&Os.

1.1.1 PP+O PROGRAMS, PRODUCTION AND OPERATIONS



4.3 OUTPUT PROVIDED BY PP&O

Distribution departments uses data for handling all MC&G inventory. Its major functions are to: produce inventory reports, receive inventory data, update inventory and maintain inventory status. Distribution departments communicate with both PPOD and PDs, in order to keep inventory data current.

4.3.2 DMIS/P Formal Specifications

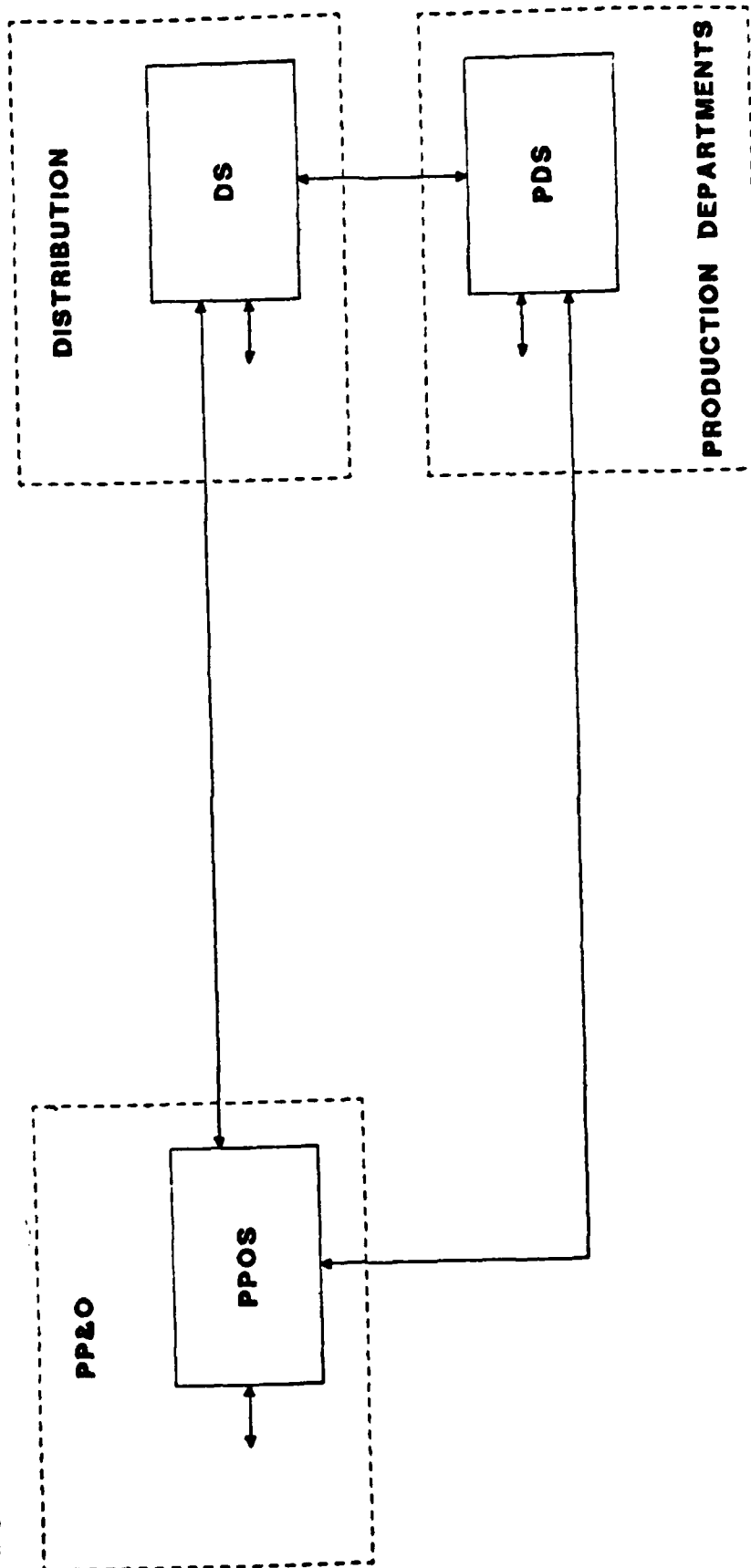
A formal description of DMIS/P is a net. The description (Called a net view), begins by naming a process in the net, followed by a description of the process. Each process in a net is described by:

- The data entities it controls and their invariant properties.
- A set of procedures acting upon the data.
- Message sending and receiving procedures associated with input and output ports.
- Rules for controlling behavior.

Procedures in a process are made up of its appropriate interfaces, input/output assertions and a return value. Neither sending or receiving procedures affect the data entities under the process control. Sending procedures accept values to be sent, and receiving procedures return some value each time it is invoked. This value is indicated by Return-Value under its proper procedure interface.

The formal description is done by first defining the data, procedures and behavior of each process, then placing these items into the framework suggested by the specification notation. The following paragraphs give this view for the three processes.

PRODUCTION, PROGRAMS
& OPERATIONS



4.4 DMIS/P NET VIEW

DISTRIBUTION SYSTEM

DATA

Inventory

PROCEDURES

- o Get Product Information
- o Update Inventory Product Information
- o Get Product Receiving Information
- o Update Inventory Receipt
- o Get Product Deliveries
- o Update Inventory Deliverables
- o Get Inventory Status Requirements from PPOS
- o Send Inventory Status to PPOS

BEHAVIOR

get.product.info ? (pdi) from PDS < update.inventory.pdi(pdi)

get.prod. receiving ? (x) < update.inv.rec. (x)

get.prod.deliver ? (x) < update.inv.deliv. (x)

get.inv.status.req ? (x) from PPOS < send.inv! (x) to PPOS

PRODUCTION DISTRIBUTION SYSTEM

DATA

- o Job Plans
- o Job Status
- o Resource Utilization
- o Digital Data

PROCEDURES

- o Get Job Plans (JP)
- o Save Job Plans
- o Get Job Plans (UPDATES)
- o Update Job Plans
- o Get Production Status Requirements
- o Send Job Status
- o Get Time Cards (TC)
- o Update Time Cards
- o Get Production Status (PDS)
- o Send Production Updates

- o Get Digital Data
- o Update Digital Inventory

CONTROL

get.job plans ? (jp) < save (jp)

get.job plan.update (jpupdt) < update.job plans (jpupdt)

get.prod.status.req ? (selidor) < send.job.status (selidor)

04076 get.time cards ? (tc) < update.ps.tc (tc)

get.product.status ? (pds) < update.ps.pds (pds)
< update.res.util. (pds)

get.product.status ? (pds) << (pds = completed MC&G product)
send.product.util (pds) to DS

get.product.status ? (pds) << (pds = completed digital product)
digit.inventory.update (pds)

PPOS

DATA

- o Resources
- o Production.Standards
- o Current.Program

PROCEDURES

get.i REQUEST ? (x)

get.requirements ? (requirements, constraints)

make plan (requirements, constraints, inventory, budget, manpower)
return plan

putout.plan ! (plan)

send req. for inventory !

get.inventory.status ? (inventory)

get.resource.updates ? (rupdt)

get.standards.updates ? (s.updt)

update.resources (r.updt)

update.standards (s.updt)

get.approved.program ? (prog)

create.program (progs)

get.program.updates ? (p.updt)

update.program (p.updt)

get.program.req. ?

put out program ! (po)

send.prod.data.req ! (sp)

get.prod.data ? (pd)

standards.revision (pd) return S

putout.standards revision (s)

get.req ? (s)

send.prod.status.req ! (s)

get.prod.status ? (pstatus)

putout.status ! (pstatus)

send job plan! (sjp) to PPS

send job plan updates ! (sjp)

BEHAVIOR

Note: These activities are related to planning.

get.requirements ? (r,c)

< send.req.for.inventory ! to Ds

< get.inventory.status ? (inv) from DS

< plan = make.plan (r,c inv)

< putout.plan ! (plan)

get.resource.updates ? (rupdt) < update.resource (r.updt)

get.standards.updates ? (s.updt) < update.standards (s.updt)

get.approved.program ? (prog) < create.program (progr.)

<send.job.plan ! () to PDS

get.program.updates ? (p.updt) < update program (p.updt)

<send.job plan.update ! () to PDS

get.request ? (x) s u = 'standards review'

send.prod.data.req ! ()

< get.prod.data ? (pd)

< s = standards.revision (pd)

< putout.standards.revision (s)

get.request ? (x) s u = 'status' < get.request ? (seledor)

< send.prod.status.req ! (seledor) to PDS

< get.prod.status ? (pstatus) from PDS

< putout.status ! (pstatus)

get.request ? (x) s u = 'program' < putout.program ! ()

NET DMIS/P

PROCESSES.

PROCESS PPOS.

DATA.

ENTITIES: Resources

 Production.Standards

 Current.Program

INVARIANT: ...

PROCEDURES.

INTERFACE: Get.Requirements ? (r,c)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: r = area requirements

 c = planning constraints

INTERFACE: p = Make.Plan (r,c,i)

INPUT-ASSERTION: r = area requirements, c = planning
 constraints, i = inventory

OUTPUT-ASSERTION: 04240

RETURN-VALUE: p = plan

INTERFACE: Putout.Plan ! (p)

INPUT-ASSERTION: p = plan

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Req.Inventory ! ()

INPUT-ASSERTION:

OUTPUT-ASSERTION

RETURN-VALUE:

INTERFACE: Get.Inventory.Status ? (i)

INPUT-ASSERTION

OUTPUT-ASSERTION

RETURN-VALUE: i = inventory

INTERFACE: Get.Resource.Updates ? (ru)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ru = updates to resource data

INTERFACE: Get.Standards.Updates ? (su)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: su = updates to standards

INTERFACE: Update.Resources (ru)

INPUT-ASSERTION: ru = updates to resource data

OUTPUT-ASSERTION: resources are updated

INTERFACE: Update.Standards (su)

INPUT-ASSERTION: su = updates to standards

OUTPUT-ASSERTION: Production.Standards are updated

RETURN-VALUE:

INTERFACE: Get.Approved.Program ? (pg)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE pg = approved program

INTERFACE: Creat.Program (pg)

INPUT-ASSERTION: pg = approved program

OUTPUT-ASSERTION: current.program is recreated

RETURN-VALUE:

INTERFACE: Get.Program.Updates ? (pu)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: pu = updates to the current program

INTERFACE: Update.Program (pu)

INPUT-ASSERTION: pu = updates to the current program

OUTPUT-ASSERTION: current.program is updated

RETURN-VALUE:

INTERFACE: Get.Request ? (x)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: x = member of approved list of requests for
 information

INTERFACE: pg = Extract.Program

INPUT-ASSERTION: current.program is available

OUTPUT-ASSERTION:

RETURN-VALUE: pg = copy of current program

INTERFACE: Putout.Program ! (pg)

INPUT-ASSERTION: rg = copy of current program

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Send.Prod.Data.Req ! ()

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Send.Prod.Status.Req ! (z)

INPUT-ASSERTION: z = valid request for production information

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Get.Prod.Status ? (ps)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ps = production status

INTERFACE: Putout.Prod.Status ! (ps)

INPUT-ASSERTION: ps = production status

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: jp = Make.Job.Plan ! ()

INPUT-ASSERTION: urrent program available

OUTPUT-ASSERTION:

RETURN-VALUE: jp = job plan data

INTERFACE: Send.Job.Plan ! (jp)

INPUT-ASSERTION: jp = job plan data

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Standards.Review.Req ! ()

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: y = Extract.Info (x)

INPUT-ASSERTION: x = member of approved list of requests for
information

OUTPUT-ASSERTION:

RETURN-VALUE: y = Extracted information from data being
maintained

INTERFACE: Send.Info ! (y)

INPUT-ASSERTION: y = Extracted information from data being

maintained

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Prod.Info.Reg ? (z)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: z = valid request for production information

BEHAVIOR.

*** Planning Activities.

Get.Requirements ? (r,c) < Req.Inventory ! () to DS
 < Get.Inventory.Status ? (i) from DS
 < p = Make.Plan (p,c,i)
 < Put.Out.Plan (p)

*** Program Maintenance.

Get.Approved.Program ? (pg) < Create.Program (pg)
 < pj = Make.Job.Plan ()
 < Send.Job.Plan ! (jp) to PDS

Get.Program.Updates ? (pu) < Update.Program (pu)
 < jp = Make.Job.Plan ()
 < Send.Job.Plan ! (jpn) to PDS

*** Resources and Standards Maintenance

Get.Resource.Updates ? (rd) < Update.Resource (ru)
Get.Standards.Updates ? (su) < Update.Standards (su)
Standards.Review.Req ? () < Send.Prod.Data.Req ! () to PDS
 < Get. Prod.Data ? (pd) from PDS
 < s = Standards.Revision (pd)
 < Put.Out.Standards ! (s)

*** Information Retrieval

Get.Request ? (x) < y = Extract.Info (x)
 < Send.Info ! (y)

Pro.Info.Req ? (z) < Send.Prod.Status.Req ! (z) to PDS
 < Get.Prod.Status ? (pdz) from PDS
 < Put.Out.Prod.Status ! (pdz)

PROCESS PDS.

DATA.

ENTITIES: Job.Plan

 Job.Status

 Resource.Utilization

 Digital.Data.Inventory

INVARIANT: ...

PROCEDURES.

INTERFACE: Get.Job.Plan ? (jp)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: jp = job plan data

INTERFACE: Update.Job.Plan (jp)

INPUT-ASSERTION: jp = job plan data

OUTPUT-ASSERTION: job.plan is updated or replaced

RETURN-VALUE:

INTERFACE: Get.Time Card ? (tc)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: tc = completed time card

INTERFACE: Update.Job.Status (ujs)

INPUT-ASSERTION: ujs = time card or product monitoring data

OUTPUT-ASSERTION: job.status is updated

RETURN-VALUE.

INTERFACE: Get.Product.Status ? (ps)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ps = product monitoring data

INTERFACE: Update.Resource.Utilization (ps)

INPUT-ASSERTION: ps = product monitoring data

OUTPUT-ASSERTION: resource.utilization is updated

RETURN-VALUE:

INTERFACE: Inform.DS ! (ps)

INPUT-ASSERTION: ps = product monitoring data indicating
completed MC&G product

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Update.Digital.Inventory (ps)

INPUT-ASSERTION: ps = product monitoring data indicating
completed digital product

OUTPUT-ASSERTION: digital.inventory is updated

INTERFACE: Get.Prod.Data.Req ? (pl)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: pd = Extract.Prod.Data (pd)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: pd = production data based on job.status and
 resource.utilization

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Get.Prod.Status.Req ? (z)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: z = valid request for production information

INTERFACE: pdz = Extract.Prod.Status (z)

INPUT-ASSERTION: z = valid request for production information

OUTPUT-ASSERTION:

RETURN-VALUE: pdz = production status in response to the
 criteria z

INTERFACE: Send.Prod.Status ! (pdz)

INPUT-ASSERTION: pdz = product status

OUTPUT-ASSERTION:

RETURN-VALUE:

BEHAVIOR.

*** Job Plan Maintenance

Get.Job.Plan ? (jp) < Update.Job.Plan (jp) from PPOS

*** Time Cards Processing

Get.Timecard ? (tc) < Update.Job.Status (tc)

*** Product Monitoring

Get.Product.Status ? (ps) < Update.Job.Status (ps)
< Update.Resource.Utilization (ps)

(ps indicates completed MC&G product) Get.Product.Status ? (ps)

<< Inform.DS ! (ps) to DS

(ps indicates completed digital product) Get.Product.Status ?

(ps) << Update.Digital.Inventory (ps)

*** Servicing PPOS

Get.Prod.Data.Req ? () < pd = Extract.Prod.Data () from PPOS
< Send.Prod.Data ! (pd) to PPOS

Get.Prod.Status.Req ? (z) < pdz = Extract.Prod.Status (z)
from PPOS < Send.Prod.Status ! (pdz) t

PROCESS DS.

DATA.

ENTITIES: MC&G.Inventory

INVARIANT ...

PROCEDURES.

INTERFACE: Info.From.PSS ? (ps)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ps = product monitoring data indicating
completed MC&G product

INTERFACE: Update.Inventory (d)

INPUT-ASSERTION: d = product monitoring data indicating
completed MC&G product or information regarding
product receiving and shipping

OUTPUT-ASSERTION: MC&G.Inventory is updated

RETURN-VALUE:

INTERFACE: Get.Inventory.Req ! (r)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: i = Extract.Inventory (i)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: i = inventory status

INTERFACE: Send.Inventory.Status ! (i)

INPUT-ASSERTION: i = inventory status

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: Get.Receiving.Data ? (x)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: x = information regarding products received

INTERFACE: Get.Shipping.Data ? (y)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: y = information regarding products being
 shipped.

4.4 Performing Hardware/Software Tradeoffs

In order to perform the hardware/software tradeoffs on the case study, the functional design developed in Section 4.3, must be specified in terms of performance attributes and constraints. The design approach begins by making some basis assumptions about performance attributes and the constraints imposed upon them. A performance evaluation model is defined which provides the means for assigning a quantitative measures to the performance attributes and constraints.

Each process in the net is evaluated in terms of its performance specifications. Using this specification as input, the steps of the methodology are applied to each process in the net. A hardware/software tradeoffs is made among system options based on the performance attributes and the constraints.

4.4.1 DMIS/P Performance Specifications

The DMIS/P has been described as a net composed of three processes, Program, Production and Operation, Distribution and Production (Figure 4.4.0). Each process in the net describes a system. For example, PP&O is supported by the system PPOS (Program, Production, and Operation Systems). The system for Distributed is DS (Distribution System), and Production Department is supported by the Production Department System (PDS). Partitioning of processes in this manner permits performance attributes and constraints to be related to each process. When assumptions are made about basic attributes and constraints developed, the design becomes a performance net. The DMIS/P Performance net is made up of:

- The data entities it controls and their invariant properties.

- A set of procedures acting upon the data.
- Messages sending and receiving procedures associated with input and output ports.
- A performance specification model consisting of:
 - Attributes
 - Constraints
 - A performance evaluation model

Since the model is made up of communicating processes, the system is assumed to be distributed. The next step in our design process is to specify the constraints, performance attributes and to relate them to a particular process with consideration for levels of distribution.

4.4.1.1 Relevant Performance Attributes

Relevant performance attributes refer to those factors that are most essential for meeting the system's performance requirements.

Based on the formal and informal specifications for DMIS/P System, the following are assumed to be the most relevant performance attributes:

- (a) Volume - Measured in number of records. (Assumed to have some size.

Volume is important because of:

- o The need to model storage.

- o The need to determine access time attributes.
- (b) Execution time - Measured in seconds: (Assumed to depend upon):
- o The nature of each procedure.
 - o The kind of hardware being considered.
 - o The volume of data being used.
- (c) Response time - measured in seconds. (Discussed in details in 4.4.1.3, due to its relevance to execution time).

4.4.1.2 Performance Evaluation Model

The performance evaluation model provides the mechanism by which derived attributes are computed from basic attributes. Fundamental assumptions of the performance evaluation model are:

- o Data access time.
- o Communication delay (considered negligible with respect to data access time).
- o The worst case situation.

The time attribute will be computed by the following formula:

$t(\text{procedure}) = k \cdot p \cdot v$ where:

k - reflects the presumed machine speed.

p - measures the relative computational complexity of the procedure.

v - measures the total data volume bounded by the procedure (this involves process data as well as input parameters and returned values).

4.4.1.3 Constraints

Response time is the attribute upon which the constraints are based. The assumption is made that response time ought to satisfy certain upper bounds.

Because of the worse case consideration, response time is measured by the sum of the execution time of all activities that could be simultaneously initiated. (All activities taking place at the same time).

4.4.1.4 Formalization of Performance Specifications

Using the same method of Section 2.2, the performance of the DMIS/P, can now be described. The three systems are formalized in the following manner.

The approach is to associate an attribute with each procedure under a process.

For example, for the process, PP&O, the attribute volume is associated with: Data, where Data defines:

(a) Resources,

(b) Production standards, and

(c) Current Program volume is assumed to be critical.

NET DMIS/P.

PROCESSES.

PROCESS PPOS.

DATA.

ENTITIES: resources Attribute vol.

 production.standards Attribute vol.

 current.program Attribute vol.

INVARIANT: ...

PROCEDURES.

INTERFACE: GET.REQUIREMENTS ? (r,c) Attribute time
 (GET.REQUIREMENTS) = (VOL(r) + vol(c)) *k

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: r = area requirements

 c = planning constraints

INTERFACE: p = MAKE.PLAN (r,c,i) Attribute time
 (MAKE.PLAN) = k * (vol(r) * vol(c) + vol(i))

INPUT-ASSERTION: r = area requirements, c = planning
constraints, i = inventory

OUTPUT-ASSERTION:

RETURN-ASSERTION: p = plan Attribute $\text{vol}(p) =$
 $\text{vol}(\text{current.program})$

INTERFACE: $\text{PUTOUT.PLAN} ! (p)$ Attribute time (PUTOUT.PLAN)
 $= k * \text{vol}(p)$

INPUT-ASSERTION: p = plan

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: $\text{REQ.INVENTORY} ! (r)$ Attribute time negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: $\text{GET.INVENTORY.STATUS} ? (i)$ Attribute time
 $(\text{GET.INVENTORY.STATUS}) = k * \text{vol}(i)$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: i = inventory Attribute vol

INTERFACE: GET.RESOURCE.UPDATES ? (ru) Attribute time
(GET.RESOURCES.UPDATES) = $k * \text{vol}(\text{ru})$

OUTPUT-ASSERTION:

RETURN-VALUE: ru = updates to resource data Attribute vol

INTERFACE: GET.STANDARDS.UPDATES ? (su) Attribute time
(GET.STANDARDS.UPDATES) = $k * \text{vol}(\text{su})$

OUTPUT-ASSERTION:

RETURN-VALUE: su = updates to standards Attribute vol

INTERFACE: UPDATE.RESOURCES (ru) Attribute time
(UPDATE.RESOURCES) = $k * \text{vol}(\text{resources}) * \text{vol}(\text{ru})$

INPUT-ASSERTION: ru = updates to resource data

OUTPUT-ASSERTION: resources are updated

RETURN-VALUE:

INTERFACE: UPDATE.STANDARDS (su) Attribute time
(UPDATE.STANDARDS) = $k * \text{vol}(\text{production.standards}) * \text{vol}(\text{su})$

INPUT-ASSERTION: su = updates to standards

OUTPUT-ASSERTION: production.standards are updated

RETURN-VALUE:

INTERFACE: GET.APPROVED.PROGRAM ? (pg) Attribute time
(GET.APPROVED.PROGRAM) = $k * vol(pg)$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: pg = approved program Attribute vol(pg) =
vol(current.program)

INTERFACE: CREATE.PROGRAM (pg) Attribute time
(CREATE.PROGRAM) = $k * vol(pg)$ =

INPUT-ASSERTION: pg = approved program

OUTPUT-ASSERTION: current.program is recreated

RETURN-VALUE:

INTERFACE: GET.PROGRAM.UPDATES ? (pu) Attribute time
(GET.PROGRAM.UPDATES) = $k * vol(pu)$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: $pu = \text{updates to the current.program Attribute}$
 vol

INTERFACE: $UPDATE.PROGRAM (pu) \text{ Attribute time}$
 $(UPDATE.PROGRAM) = k * vol (\text{current.program}) *$
 $vol(pg)$

INPUT-ASSERTION: $pu = \text{updates to the current.program}$

OUTPUT-ASSERTION: $\text{current.program is updated}$

RETURN-VALUE:

INTERFACE: $GET.REQUEST ? (x) \text{ Attribute time Negligible}$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: $x = \text{member of approved list of requests for}$
 information

INTERFACE: $GET.INVENTORY.STATUS ? (i) \text{ Attribute time}$
 $(GET.INVENTORY.STATUS) = k * vol(i)$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: $i = \text{inventory Attribute vol}$

INTERFACE: GET.RESOURCE.UPDATES ? (ru) Attribute time
(GET.RESOURCES.UPDATES) = k * vol(ru)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ru = updates to resource data Attribute vol

INTERFACE: GET.STANDARDS.UPDATES ? (su) Attribute time
(GET.STANDARDS.UPDATES) = k * vol(su)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: su = updates to standards Attribute vol

INTERFACE: UPDATE.RESOURCES (ru) Attribute time
(UPDATE.RESOURCES) = k * vol(resources *
vol(ru))

INPUT-ASSERTION: ru = updates to resource data

OUTPUT-ASSERTION: resources are updated

RETURN-VALUE:

INTERFACE: UPDATE.STANDARDS (su) Attribute time
(UPDATE.STANDARDS) = k * vol
(production.standards) * vol(su)

INPUT-ASSERTION: su = updates to standards

OUTPUT-ASSERTION: production.standards are updated.

RETURN-VALUE:

INTERFACE: pg = EXTRACT.PROGRAM Attribute time
(EXTRACT.PROGRAM) = $k * \text{vol}(\text{current.program})$

INPUT-ASSERTION: current.program is available

OUTPUT-ASSERTION:

RETURN-VALUE: pg = copy of current program
 $\text{vol}(\text{pg}) = \text{vol}(\text{current.program})$

INTERFACE: PUT.OUT.PROGRAM ? (pg) Attribute time
(PUT.OUT.PROGRAM) = $k * \text{vol}(\text{pg})$

INPUT-ASSERTION: pg = copy of current program

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: SEND.PROD.DATA.REQ ? (pd) Attribute time
Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: GET.PROD.DATA ? (pd) Attribute time
(GET.PROD.DATA) = k * vol(pd)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: pd = production data Attribute vol

INTERFACE: s = STANDARDS.REVISION (pd) Attribute time
(STANDARD.REVISION) = k * vol(pd) *
vol(production.standards)

INPUT-ASSERTION: pd = production.data

OUTPUT-ASSERTION: production.standards are revised based on new
production

RETURN-VALUE: s = revised production.standards Attribute
vol(s) = vol(production.standards)

INTERFACE: PUT.OUT.STANDARDS ? (s) Attribute time
(PUT.OUT.STANDARDS) = k * vol(s)

INPUT-ASSERTION: s = production standards copy

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: SEND.PROD.STATUS.REQ ! (z) Attribute time
Negligible

INPUT-ASSERTION: z = valid request for production information

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: GET.PROD.STATUS ? (ps) Attribute time
(GET.PROD.STATUS) = k * vol(ps)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ps = production status Attribute vol

INTERFACE: PUT.OUT.PROD.STATUS ! (ps) Attribute time
(PUT.OUT.PROD.STATUS) = k * vol(ps)

INPUT-ASSERTION: ps = production status

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: jp = MAKE.JOB.PLAN () Attribute time
(MAKE.JOB.PLAN) = k * vol(current.plan)

INPUT-ASSERTION: current program available

OUTPUT-ASSERTION:

RETURN-VALUE: $jp = \text{job plan data Attribute vol}(jp) =$
 $\text{vol}(\text{current.plan})$

INTERFACE: $\text{SEND.JOB.PLAN ! (jp) Attribute time}$
 $(\text{SEND.JOB.PLAN}) = k * \text{vol}(jp)$

INPUT-ASSERTION: $jp = \text{job plan data}$

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: $\text{STANDARDS.REVIEW.REQ ? () Attribute time}$
 Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: $y = \text{EXTRACT.INFO}(x) \text{ Attribute time}$
 $(\text{EXTRACT.INFO}) = k * \text{vol}(\text{resources}) +$
 $\text{vol}(\text{production.standards}) +$
 $\text{vol}(\text{current.program})/100$

INPUT-ASSERTION: $x = \text{member of approved list of requests for}$
 information

OUTPUT-ASSERTION:

RETURN-VALUE: $y = \text{extracted information from data being}$
 $\text{maintained Attribute } \text{vol}(y) = \text{time}$
 $(\text{EXTRACT.INFO}) / (10 * k)$

INTERFACE: $\text{SEND.INFO ? (y) Attribute time (SEND.INFO) = k}$
 $* \text{vol}(y)$

INPUT-ASSERTION: $y = \text{extracted information from data being}$
 maintained

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: $\text{PROD.INFO.REQ ? (z) Attribute time Negligible}$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: $z = \text{valid reques for production information}$

BEHAVIOR.

*** Planning Activities

GET.REQUIREMENTS ? (r,c) < REQ.INVENTORY ! () to DS
 < GET.INVENTORY.STATUS ? (i) from
 < p = MAKE.PLAN (p,c,i)
 < PUT.OUT.PLAN (p)

*** Program Maintenance.

GET.APPROVED.PROGRAM ? (pg) < UPDATE.PROGRAM (pu)
 < jp = MAKE.JOB.PLAN (jp)
 < SEND.JOB.PLAN ! (jp)

GET.PROGRAM.UPDATES ? (pu) < UPDATE.PROGRAM (pu)
 < pjn = MAKE.JOB.PLAN (jp)
 < SEND.JOB.PLAN ! (jpn) to P

***Resources and Standards Maintenance.

GET.RESOURCES.UPDATE ? (ru) < UPDATE.RESOURCES (ru)

GET.STANDARDS.UPDATES ? (su) < UPDATE.STANDARDS (su)

STANDARDS.REVIEW.REQ ? () < SEND.PROD.DATA.REQ ! (sp) to
 < GET.PROD.DATA ? (pd)
 < s = STANDARDS.REVISION (pd)
 < PUT.OUT.STANDARDS ! (s)

*** Information Retrieval.

GET.REQUEST ? (x) < y = EXTRACT.INFO (x)
 < SEND.INFO ! (y)

PROD.INFO.REQ ? (z) < SEND.PROD.STATUS.REQ ! (z)
 < GET.PROD.STATUS ? (pdz)
 < PUT.OUT.PROD.STATUS ! (pdz)

PROCESS PDS.

DATA.

ENTITIES: job.plan Attribute vol(job.plan) = vol(current.program
in PPOS)

job.status Attribute vol

resource.utilization Attribute vol

digital.data.inventory Attribute vol

INVARIANT: ...

PROCEDURES.

INTERFACE: GET.JOB.PLAN ? (jp) Attribute time
(GET.JOB.PLAN) = $k * vol(jp)$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: jp = job.plan.data

INTERFACE: UPDATE.JOB.PLAN (jp) Attribute time
(UPDATE.JOB.PLAN) = $k * vol(jp)/100$

INPUT-ASSERTION: jp = job.plan.data

OUTPUT-ASSERTION: job.plan is updated or replaced

RETURN-VALUE:

INTERFACE: GET.TIMECARD ? (tc) Attribute time Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: tc = completed time card

INTERFACE: UPDATE.JOB.STATUS (uj) Attribute time
(UPDATE.JOB.STATUS) = k * vol(job.status)/100

INPUT-ASSERTION: uj = time card or product monitoring data

OUTPUT-ASSERTION: job.status is updated

RETURN-VALUE:

INTERFACE: GET.PRODUCT.STATUS ? (ps) Attribute time
Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ps = product monitoring data

INTERFACE: UPDATE.RESOURCE.UTILIZATION (ps) Attribute time

(UPDATE.RESOURCE.UTILIZATION) = k' *
vol(resource.utilization)/10

INPUT-ASSERTION: ps = product monitoring data

OUTPUT-ASSERTION: resource.utilization is updated

RETURN-VALUE:

INTERFACE: INFORM.DS ! (ps) Attribute time Negligible

INPUT-ASSERTION: ps = product monitoring data indicating
completed MC&G product

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: UPDATE.DIGITAL.INVENTORY (ps) Attribute time
(UPDATE.DIGITAL.INVENTORY) = k' *
vol(digital.data.inventory)/10

INPUT-ASSERTION: ps = product monitoring data indicating
completed digital product

OUTPUT-ASSERTION: digital.inventory is updated

RETURN-VALUE:

INTERFACE: GET.PROD.DATA.REQ ? () Attribute time
Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: $pd = \text{EXTRACT.PROD.DATA} () \text{ Attribute time}$
 $(\text{EXTRACT.PROD.DATA}) = k' * \text{vol}(\text{job.status}) +$
 $\text{vol}(\text{resource.utilization})$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: $pd = \text{production data based on job.status and}$
 $\text{resource.utilization Attribute time vol(po) =}$
 $10 * \text{vol(pds)}$

INTERFACE: $\text{SEND.PROD.DATA ! (pd) Attribute time}$
 $(\text{SEND.PROD.DATA}) = k' * \text{vol(po)}$

INPUT-ASSERTION: $pd = \text{product data based on job.status and}$
 $\text{resource.utilization}$

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: $\text{GET.PROD.STATUS.REQ ? (z) Attribute time}$
 Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: $z = \text{valid request for production information}$

INTERFACE: $\text{pdz} = \text{EXTRACT.PROD.STATUS}(z) \text{ Attribute time}$
 $(\text{EXTRACT.PROD.STATUS}) = k' * \text{vol}(\text{job.status}) +$
 $\text{vol}(\text{resource.utilization})/10$

INPUT-ASSERTION: $z = \text{valid request for production information}$

OUTPUT-ASSERTION:

RETURN-VALUE: $\text{pdz} = \text{production status in response to the}$
 $\text{criteria } z \text{ Attribute vol}$

INTERFACE: $\text{SEND.PROD.STATUS} ! (\text{pdz}) \text{ Attribute time}$
 $(\text{SEND.PROD.STATUS}) = k' * \text{vol}(\text{pdz})$

INPUT-ASSERTION: $\text{pdz} = \text{production status}$

OUTPUT-ASSERTION:

RETURN-VALUE:

BEHAVIOR.

*** Job Plan Maintenance.

GET.JOB.PLAN ? (jp) < UPDATE.JOB.PLAN (jp) from PPOS

*** Time Cards Processing.

GET TIMECARD ?(tc) < UPDATE.JOB.STATUS (tc)

*** Product Monitoring.

GET.PRODUCT.STATUS ? (ps) < UPDATE.JOB.STATUS (ps)

< UPDATE.RESOURCE.UTILIZATION (ps)

(ps indicates completed MC&G product)GET.PRODUCT.STATUS ?

(ps) <<INFORM.DS !

(ps indicates completed digital product)GET.PRODUCT.STATUS ? (ps)

<<UPDATE.DIGITAL.INVENTORY (ps)

*** Servicing PPOS

GET.PROD.DATA.REQ ? (jp) < pd = EXTRACT.PROD.DATA (jp)

< SEND.PROD.DATA ! (pd) to PPOS

GET.PROD.STATUS.REQ ? (z) < pdz = EXTRACT.PROD.STATUS (z) from

< SEND.PROD.STATUS ! (pdz)

AD-A093 680 TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CA
FAST METHODOLOGY & CASE STUDY. (U)
NOV 80 F306

TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CA
FAST METHODOLOGY & CASE STUDY.(U)
NOV 80 F306

F/6 9/2

UNCLASSIFIED

RADC-TR-80-336

F30602-79-C-0078

NL

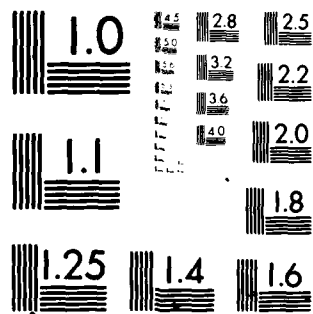
3 OF 3

END

DATE _____

FILED
2-84

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

PROCESS DS.

DATA.

ENTIRIES: MC&G inventory Attribute vol

INVARIANT: ...

PROCEDURES.

INTERFACE: INFO.FROM.PDS ? (ps) Attribute time Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: ps = product monitoring data indicating
completed MC&G product

INTERFACE: UPDATE.INVENTORY (d) Attribute time
(UPDATE.INVENTORY) = k' *
vol(MC&G.INVENTORY)/100

INPUT-ASSERTION: d = product monitoring data indicating
completed MC&G product or information regarding
product receiving and shipping

OUTPUT-ASSERTION: MC&G.INVENTORY is updated

RETURN-VALUE:

INTERFACE: GET.INVENTORY.REQ ! (r) Attribute time
Negligible

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: i = EXTRACT.INVENTORY (i) Attribute time
(EXTRACT.INVENTORY) = k' * vol(MC&G.INVENTORY)

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: i = inventory status Attribute vol(i) =
vol(MC&G.INVENTORY)/5

INTERFACE: SEND.INVENTORY.STATUS ! (i) Attribute time
(SEND.INVENTORY.STATUS) = k' * vol(i)

INPUT-ASSERTION: i = inventory status

OUTPUT-ASSERTION:

RETURN-VALUE:

INTERFACE: GET.RECEIVING.DATA ? (x) Attribute time
(GET.RECEIVING.DATA) = 10 * k'

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: x = information regarding products received

INTERFACE: GET.SHIPPING.DATA ? (y) Attribute time
 (GET.SHIPPING.DATA) = $10 \cdot k'$

INPUT-ASSERTION:

OUTPUT-ASSERTION:

RETURN-VALUE: y = information regarding products being
 shipped

BEHAVIOR.

*** Product Availability Data.

INFO.FROM.PDS ? (ps)

< UPDATE.INVENTORY (ps) from

*** Servicing PPOS.

GET.INVENTORY.REQ ? (r)

< i = EXTRACT.INVENTORY (k) from

< SEND.INVENTORY.STATUS ! (i)

*** Housekeeping.

GET.RECEIVING.DATA ! (x)

< UPDATE.INVENTORY (x)

GET.SHIPPING.DATA ! (y)

< UPDATE.INVENTORY (y)

LINKS.

REQ.INVENTORY ! () PPOS to GET.INVENTORY.REQ ? () DS

SEND.JOB.PLAN ! () PPOS to GET.JOB.PLAN ? () PDS

SEND.PROD.DATA.REQ ! () PPOS to GET.PROD.DATA.REQ ? () PDS

SEND.PROD.STATUS.REQ ! (z) PPOS to GET.PROD.STATUS.REQ ? (z)
PDS

INFORM.DS ! (ps) PDS to INFO.FROM.PDS ? (ps) DS

SEND.PROD.DATA ! (pd) PDS to GET.PROD.DATA ? (pd) PPOS

SEND.PROD.STATUS ! (pdz) PDS to GET.PROD.STATUS ? (pdz) PPOS

SEND.INVENTORY.STATUS ! (i) DS to GET.INVENTORY.STATUS ? (i)
PPOS

COMMUNICATION.

* ! (*) << * ? (*)

Note: Each message sending is followed by the corresponding
message receiving without message loss.

Constraints

"Worst case response time, W, must satisfy the condition that

$$W < 60 \text{ sec.}$$

Worst case response time is defined as the total execution time of all sequence of activities that could be initiated simultaneously.

ASSUMPTIONS

PPOS: vol(resource) = 500 (basic-validated)

vol(production-standards) = 500 (basic-validated)

vol(current programs) = 4000 (basic-validated)

vol(r) = 1000 (basic-validated)

vol(c) = 100 (basic-validated)

vol(ru) = 10 (basic-validated)

vol(su) = 10 (basic-validated)

vol(pu) = 10 (basic-validated)

PDS: vol(job.status) = 1000 (basic-validated)

vol(resource.utilization) = 500 (basic-validated)

vol(digital.data.inventory) = 1000 (basic-validated)

$\text{vol}(\text{pdz}) = 100$ (basic-validated)

DS: $\text{vol}(\text{mc\&g inventory}) = 10000$ (basic-validated)

Given the basic-validated attributes above, it is now possible to compute derived attributes as a function of the basic-assumed attributes (K, K' and K").

Finally a worst case response time can be computed for each of the three processes independently.

PPOS: $t = 200180K$

PDS: $t' = 7400K'$

DS: $t'' = 12120 K''$

The value of w(worst case response) depends upon the partitioning of the three processes. Maximum value for w is incurred when all three processes are viewed as sharing a single physical machine:

$$K = K' = K''$$

$$W = 219700 K \text{ which requires } K < .273 \text{ ms.}$$

Maximum concurrency results is

$$W = \max(200180K, 7400K', 12120K'') \text{ with } K < .299\text{ms. } K' < 8.1\text{ms.}$$

$$K'' < 4.9 \text{ ms.}$$

4.4.2 Step by Step Application of FAST

The performance specification model developed in the previous section provides the input needed to perform the hardware/software tradeoffs. The model tells exactly what the system is to do; and assigns quantitative measures to the assumptions and constraints. The DMIS/P performance specification model has the following characteristics:

- (1) The system is distributed consisting of three processors.
- (2) The processors communicate with each other by sending and receiving messages.
- (3) Relevant performance attributes are: volume, size, and time
- (4) System behavior (the kinds of transactions, the number of users, response time, software requirements, etc.).

The system development process proceeds by taking their specifications and applying every step of the methodology to it. This system specification is discussed in Section 2.1.5, and is defined as a machine independent processing model. Its unique attributes are data, messages, event, and processor. No attempt is made at this point to assign physical machines or components to the model. Key activities are to:

- (a) Establish units of distribution with system processors, and
- (b) Describe the system's functions so as to allow design options to vary. Each step of the methodology is now applied to this specification in order to make hardware/software tradeoffs

among the available options.

4.4.2.1 Identification of Distribution Units

- o There are no restrictions regarding distribution.
- o If possible all three processes will be put on a single machine.
- o If necessary each process may be placed on separate machines. (Determined by the amount of work to be performed by each processor).

4.4.2.2 Generation of Hardware/Software Partitioning Constraints

- o Local high speed communications is assumed within DMAAC.
- o User imposed response time on worst case needs to be less than 60 sec.
- o Data volume as described in Section 4.4.1.4 must be satisfied by disk space (design originated constraint).

4.4.2.3 Selecting Hardware/Software Partitioning Option

- o Off-the-shelf minicomputers with customer and vendor application software and data base is considered.

Reason:

- o Most likely option that would keep price down.

- o Only one option is treated due to time availability on the project.

4.4.2.4 Development of Hardware/Software Partitioning Rules

- o A computer selection methodology is proposed under this step.
- o The step is not relevant to the case study due to the fact that a single machine will be considered. In this situation the selection of the machine is abstract rather than market availability.

4.4.2.5 Selection of Competing Candidate Partitions

A machine model is defined where $M = (\text{Disk-space}, \text{Disk-asses-time}, \text{System-overhead})$ The following arbitrary characteristics are assumed:

Disk space = 30,000 (this is the same unit as in the performance specifications)

Disk-access-time = 0.23 ms

System-overhead = 20%

4.4.2.6 Candidate Evaluation

Candidate evaluation is reduced to a performance check between the attributes of the performance specifications and those of the hypothetical machine.

Consideration is given to:

- (a) Storage Requirements

Total storage need of all three processors are:

PPOS: less than 25,000

PDS: less than 10,000

DS: less than 15,000 06069

The above criteria suggest the use of at least two machines, for PPOS, and one for DS and PDS.

(b) Response Time

Assuming the above partitioning, worst case response time becomes

$$W = \text{Max}(200180K, 19520K') \text{ with } K < .299\text{ms and } K' < 3.073\text{ms}$$

These assumptions are feasible and can be meet because:

$$\text{Disk-access-time} * \text{System-overhead} = .28\text{ms} < .29$$

Binding of process is successful.

4.4.2.7 Elimination of Incongruent Configurations

The purpose of this step is to eliminate configurations that are incongruent and conflict with each other. The result of the binding of the case study was only homogeneous configuration with no incongruencies.

4.4.2.8 Communication Medium Binding

High speed communication lines is assumed. The baudwidth must be such that it makes communication negligible relative to disk access time.

4.4.2.9 Assignment of Cost/Value Coefficient

This step is not relevant for the cast study since no real machines are selected.

4.4.2.10 Selection of Winning Configuration

This step is also not relevant since no real machines are selected. The final product (see Figure 4.4), that specifies:

- (a) The kind of configuration,
- (b) The cost, and
- (c) User requirements, and
- (d) Risk associated with implementing the system.

4.5 Discussion

This section has examined the potential use and effectiveness of the FAST methodology by applying its steps to a single case study. A major activity of the methodology was to make some basic assumptions about the proposed system, to evaluate these assumptions in terms of constraints and make hardware/software tradeoffs. Additionally, the activity hopefully would point out areas of weakness, potential risk in implementing the methodology and additional tools and techniques needed to develop and fully implement the methodology.

The hardware/software analysis began by receiving as input a formal specification of the DMIS/P system. Certain assumptions were made about the system, and performance attributes and constraints were identified. These were referred to as relevant performance attributes and for this study, were identified as: volume, execution time and response time. Relevant performance attributes established base on system needs that are most important in terms of its overall mission.

The second major activity in performing the hardware/software tradeoffs, was identifying a performance evaluation model. The model proposed to answer questions such as what is a favorable and worst case situation that can be tolerated by the system. For example, in the process, PPOS: For a given procedure interface: GET REQUIREMENTS. Using the formula specified by the performance evaluation we get:
$$\text{time}(\text{GET REQUIREMENTS}) = (\text{vol}(r) + \text{vol}(c)) * K.$$
 Using some arbitrary values for the attributes r , and c , we get: $t(\text{GET REQUIREMENTS}) = (1000 + 100) * K = 1100K$. The computation is complete when for every process, and every procedure in that process, a computed value is made on some attributes. A sum for each process can now be compared with a worst case situation. This establishes a system of constraints. In the case of PPOS, $t = 200180K$, K , K' and K'' is computed for each of the three

processes. This kind of evaluation can be done for an infinite number of such processes, thus providing systems flexibility.

Finally, using the evaluation model, a mathematical value can be computed for any number of performance attributes and a hardware/software tradeoff made among those that are applicable to the constraint, "worst case" situation.

In Section 4.4.2.3, a hardware/software is made because; evaluation criteria suggest a response time of no less than 60 sec, a total storage requirement for each of the three processes as: $(25000 + 10000 + 15000)$, etc. There are many real machines on the market meeting these constraints. The tradeoff simply reduces to a computer selection methodology for these given situations. The methodology recommends the use of simulation and critical benchmark techniques to aid in the selection process.

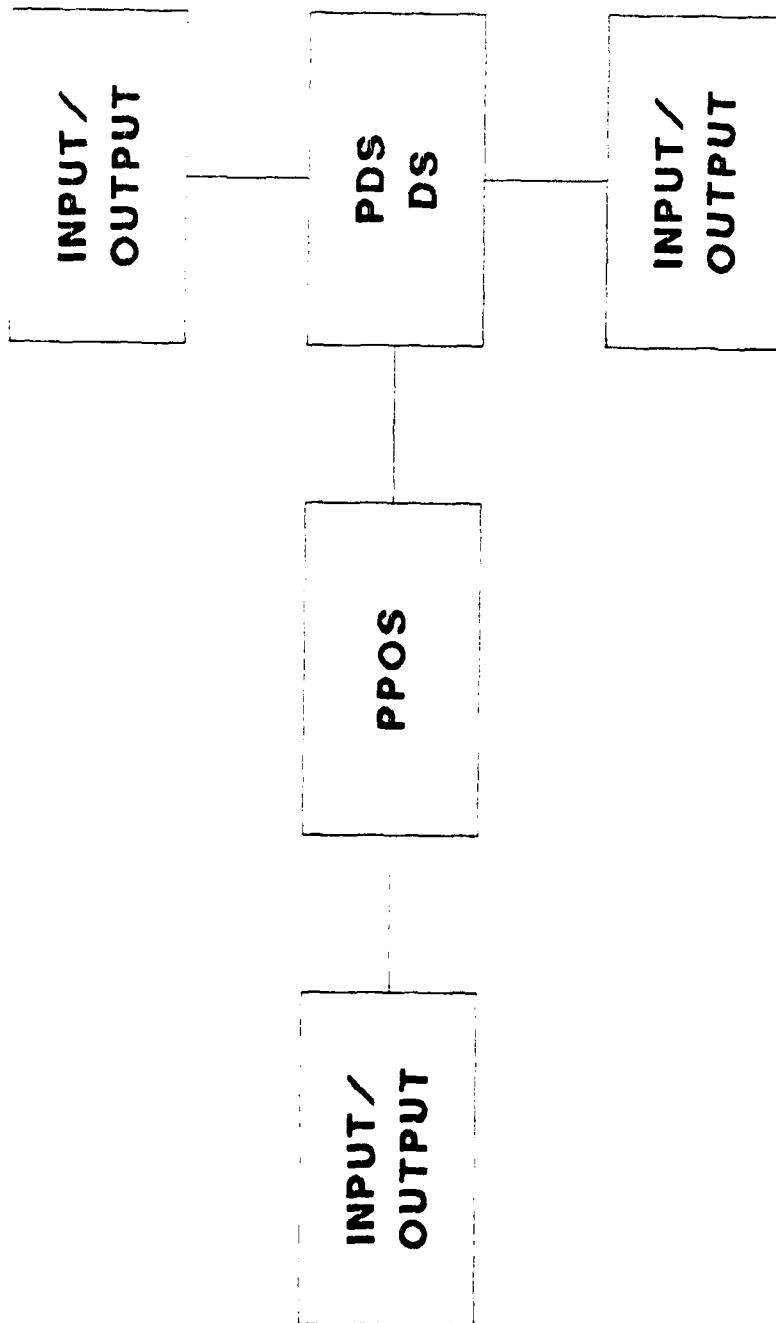


FIGURE 4.5 DMIS/P SYSTEM CONFIGURATION

The approach is applicable to a wide range of system options. Suppose for example, it was found that no commercial-off-the-shelf system was available meeting these requirements. The strategy would be to consider the next option (customized-machines).

Finally, several iterations through the system should give an optimal configuration. Constraints and attributes may be varied, as system objectives change. The above instance through the case study suggest the following minimum system configuration.

A distributed process consists of three nodes (one node for each of the three processes: PPOS, PDS, and DS).

A distributed data base shared by PDS and DS, controlled by PPOS).

NODE CONFIGURATION

For PPOS:

- o minicomputer system 1M
- o 10 CRT Terminals
- o 30 MB of Mass Storage
- o 1 medium Speed Printer (100 - 300 lpm)
- o 2 High Speed Printers (3000 lpm)
- o Transaction/Interactive software

For PDS

- o One Microprocessor 512
- o 10 MB Mass Storage
- o 5 CRT Terminals
- o 1 High Speed Printer (3000 1pm)
- o 2 Medium Speed Printers
- o Communication Equipment
- o Transaction/Interactive Software

For DS

- o 1 Microprocessor 128KB
- o 10 MB of Mass Storage
- o One medium Speed Printer
- o Communication Equipment
- o Transaction/Interactive Software

A view of this configuration is given in Figure 4.5

What general conclusions can be made about the methodology based on performing the hardware/software tradeoffs for the DMIS/P Case Study? It appears that the following conclusions may be made about the

methodology:

- (1) A clearly defined method for forming the performance specification model. The methodology should address the issue of variability in performance specifications. As options change, there should be some way to measure tradeoffs in terms of performance and constraints.
- (2) Identification of distribution units may prove to be a non-trivial task, especially, when system constraints have not been clearly defined.
- (3) Generation of the hardware/software partitioning constraints presents some risk. Exactly how these constraints are partitioned were not completely answered by this case study activity. The main reason being the limited number of constraints and performance attributes.

Several iterations through this case study is needed before any conclusions can be drawn about the real use of the methodology.

5.0 CONCLUSIONS

This paper has discussed a definition, goals, methodology steps and tools for performing a hardware/software tradeoffs. Major task were:

- (1) Developed a framework upon which to build a hardware/software tradeoffs methodology.
- (2) Defined steps for performing a hardware/software tradeoffs analysis.
- (3) Evaluated the steps in a DMA cast study.

The above activities identified a plethora of issues that should be considered in a hardware/software tradeoffs analysis. Some of the more significant ones were:

- (1) Formalization of the role played by hardware/software tradeoffs in the overall system development cycle. This has been accomplished through the use of the Total System Development Methodology Framework.
- (b) Identification of the key hardware/software tradeoffs options. It has been determined that hardware/software tradeoffs expand over a significantly wider spectrum than previously anticipated. At one extreme, there is partitioning between custom software and commercial software and hardware. At the other extreme, there is partitioning of system functions between VLSI custom designed chips and inter-chip communication.
- (c) Consolidation of functional and performance system

specifications. Hardware/software tradeoffs analysis presupposes the existence of certain performance goals that need to be reached. They may be supplied as constraints over performance attributes attached to the functional specification of the system. A specification language that has the ability to define attributes and constraints has been proposed, (1) for use in the case study and (2) for achieving a better understanding with regards to the manner in which a system's functionality and performance need to be specified when supplied as input to hardware/software tradeoffs analysis. Many of the proposed language features should be evaluated for potential adoption of some already existing specification language such as RSL.

- (d) Development of a general strategy to hardware/software tradeoffs analysis.

The feasibility of the proposed hardware/software tradeoffs methodology has been evaluated in two different ways during the performance of the contract. First, the methodology demonstrates appropriate usage of diagnostic emulation facility supported by SMITE in performing hardware/software tradeoffs. (The option being considered is a partitioning between custom software and a custom minicomputer). Second, the methodology was used to meet the specific needs of a DMA based case study where the option involved was partitioning between custom software and off-the-shelf hardware.

It is our firm belief that this study, provides a solid base for the systematic development of techniques and automated tools that will aid in the hardware/software tradeoffs analysis for a large variety of classes of application problems. In addition to identifying what has to be done in each step and the way steps should be sequenced, the study also attempted to indicate the techniques and tools available to support

each step. Our research has shown that; there are still many tools needed; and additional techniques for the different options.

Consequently, we recommend that future work in this area be separated into several studies. Each study considering in depth, one single option and focusing on:

- (1) Development of techniques and proposal of automated tools specific for that option and not yet available.
- (2) Integration of the new techniques with the already existing technology.
- (3) Investigation of ways to achieve maximum productivity with the techniques considered in (1) and (2) above be minimizing either the number or the complexity of the iterations required to obtain an acceptable solution.

Each study should be concluded by an appropriate DoD significant case study. While there is no need to plan to perform one such study for every option, a broad spectrum should be covered. For instance, one may consider custom-software/off-the-shelf hardware and a case study involving a large C(3) or distributed data base system; another could be a follow-up on the current experience with diagnostic emulation and the custom-software/custom-mini option; and still another could be dedicated to the emerging area of VLSI design and its very different hardware/software tradeoffs characters. Special architectures of interest to DoD could also be considered.

Upon completion of several of these studies a new integration effort should be contemplated. Its aim should be to propose a plan for bringing together the respective techniques under the umbrella of a single unified facility. Besides supporting hardware/software tradeoffs

analysis, the facility could aid the phases preceding (e.g., requirements definition or conceptualization) and following (e.g., software and/or hardware design) the tradeoffs performance.

Finally, our research strongly indicates that attention should be given to the emerging area of functional/performance specification of distributed systems. Because the hardware/software tradeoffs analysis presupposes the existence of a description of system's functionality and performance objectives, any advances in this area are expected to impact the effectiveness of the methodology being employed. Moreover, in many cases, the ability to: (1) specify performance attributes, constraints, and the performance evaluation model being used, and (2) relate this information to some machine models, is fundamental to the performance of hardware/software tradeoffs. (This occurs, for instance, when a hardware description language is unsuitable because of the high level of the specification or the complexity of the machine). Nevertheless, there is little that we know today about the nature of the attributes, constraints, performance models, and machine models appropriate for use with particular options. Neither is there available any systematic way to relate the system characteristics as reflected at one level of the specification to the next. The specification language used on the case study illustrates some of the issues identified above and give some indication of the direction one might consider. Work in the functional/performance specification areas could be considered both within a stand alone investigation or in conjunction with the other studies suggested earlier in the discussion.

REFERENCES

1. Alford, Mack W., "A Requirements Engineering Methodology for Real-Time Processing Requirements", IEEE Transactions on Software Engineering, Vol. SE-3, #1, Jan. 1977 pp. 60-69.
2. Alford, M. W. and I. F. Burns, "An Approach to Stating Real-Time Processing Requirements", Presented at Conf. Petri Nets and Related Methods, Massachusetts Inst. Technol., Cambridge, MA, July 1-2, 1975.
3. Alford M. W. and I. F. Burns, "R-Nets: A Graph Model for Real-Time Software Requirements", Presented at MRI Conf. Software Engineering, New York, NY, Apr. 1976.
4. Anderson, S E. and G. E. Short, "A Study of Automated Aids for Secure Systems," TRW-SS-74-06, June 1974.
5. M. R. Baracci and R. A. Parker, "Using Emulation to Verify Formal Machine Descriptions", COMPUTER, Vol. 11, No. 5, May 1978.
6. M. R. Barbacci, W. D. Dietz, and L. J. Szowerenko, "Specification, Evaluation and Validation of Computer Architectures Using Instruction Set Processor Descriptions", ACM/IEEE 1979 International Symposium on Computer Hardware Description Languages and Their Applications. October 1979, Palo Alto, California.
7. Bell, T. E., D. C. Bixler, and M. E. Dyer, "An Extendible Approach to Computer-Aided Software Requirements Engineering", TRW Software Series Number TRW-SS-76-05, July 1976.,
8. Bell, G. G. Mudge, J. C. and McNamara, J. E., "Computer Engineering: A DEC View of Hardware Systems Design Digital Press". Digital Equipment Corporation, 1978.

9. Bell C. G. and A. Newell, "Computer Structures: Readings and Examples", McGraw-Hill Book Company, New York, 1971.
10. B. W. Boehm, "Software and Its Impact: A Quantitative Assessment", DATAMATION, Vol. 19, 48-59, May 1973.
11. Boehm B. W., "Software Reliability - Measurement and Management", Abridged Proc. Software Management Conference. AIAA, Los Angeles, CA 1976.
12. Boehm B. W. , R. K. McClean, and D. B. Urfrig, "Some Experience with Automated Aids in the Design of Large Scale Reliable Software". Proc. 1975 International Conference on Reliable Software. IEEE, New York, 1975, pp. 105-113.
13. Boehm B. W., "Some Information Processing Implications of Air Force Space Missions, 1970-1980". The Rand Corporation Report, p-4947, January 1970.
14. Boyse, J. W., and Warm, D. R., "A Straightforward Model for Computer Performance Prediction". Computing, Surveys 7, (June 1975).
15. Brooks, F. P., "The Mythical Man-Month", Addison-Wesley Publishing Co., Reading, Mass. 1975.
16. Bucci, G., and Streeter, D. N., "A User-Oriented Approach to the Design of Distributed Information Systems". In Measuring, Modelling and Evaluating Computer Systems, H. Beilner, and E. Gelenbe, Eds., North-Holland Pub. Co., Amsterdam. 1977.
17. Bucci, G., and D. J. Streeter "A Methodology for the Design of

Distributed Information Systems", Communication of ACM, Vol. 22,
#4 April, 1979, pp. 233-244.

18. Burr, W. E., Coleman, A. H., and Smith, W. R., "Overview of the Military Computer Family Selection", Proceedings. 1977 National Computer Conference. AFIP Press, Montvale, N. J. 1977, pp. 131-138.
19. Clark, Bruce, Capt. N. and 2Lt. Michael A. Troutman, USAF Rome Air Development Center, Griffiss AFB, NY. The System Architecture Evaluation Facility, An Emulation Facility at Rome Air Development Center, Griffiss AFB, NY.
20. Clark, Bruce, Capt. N., "The Total System Design Methodology", Rome Air Development Center, Griffiss Air Force Base, NY, Summer, 1978, pp. 1-10.
21. Denning, Peter J., "Third Generation Computer Systems", Computing Surveys, December 1971, pp. 175-216.
22. DeRoze, B. C., "DOD Software Management Program Implementation Status and Actions", Abridged Proc., Software Management Conference, AIAA, Los Angeles, CA. 1977.
23. Freeman, H. A., "System Design Methodology - A First Step", IEEE Transactions on Software Engineering. pp. 213-215.
24. Fucik, George, "Automated Design of Distributed Special Purpose Processors", 1st International Conference on Distributed Computer Computing Systems, Oct. 1979.
25. Hansen, P. B., "Concurrent Programming Concepts", Computer Surveys, 5 (Dec 1973) No. 4 pp. 223-245.

26. Hamilton, M. and Zeldin, S., "Higher Order Software - Defining Software", IEEE Transactions on Software Engineering, SE-2 #1 March, 1976.
27. Jackson, Alyce, "The FAST Methodology: An Interim Technical Prepared under Contract to RADC", Oct. 1979.
28. Jensen, R. W. and Tonies C. C., "Software Engineering", Prentice-Hall, 1979
29. Kobayashi, H., "Modeling and Analysis - An Introduction to System Performance Evaluation". Addison-Wesley Publishing Co., California 1979.
30. Kodres, U. R., "Analysis of Real-Time Systems of Data Flow Graphs", IEEE Transactions on Software Engineering, May, 1978, Vol. SE-4, #3, 169-178.
31. Levin, D., "Computer Aided Design of Digital Systems", Crane Russak and Company, 1977.
32. Mariani, M. P., "Baseline Design Approach", Phase II of Distributed Processing Architecture Design (DPAD) Programs, Vol. III Technical Report, TRW, Feb. 1979.
33. Mesarevic, Mihajlo, and others, "Views on General Systems Theory". Proceedings of the Second Systems Symposium at Case Institute of Technology, John Wiley, New York, 1964, pp. 61-88.
34. McClean, R. R. and B. Press, "The Flexible Analysis Simulation and Test Facility: Diagnostic Emulation", Oct. 1975 TRW-SS-75-03 pp. 15-17.

35. Myers, Glenford, "Composite/Structure Design", Van Nostrand Reinhold Company, New York, 1978.
36. Myers, W., "The Need for Software Engineering", Computer Vol. 11, No. 2, pp. 13-27, Feb. 1978.
37. Myers, W., "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP85)", SAMSO TR 72-122, 1972-73.
38. Ostrofsky, Benjamin, "Design, Planning and Development Methodology". Prentice-Hall, New Jersey, 177 pp. 45-66.
39. Palk, G., "A Comparison of Network Architectures-The ARPANET and SNA", Proceedings AFIPS National Computer Conference, 1978.
40. Parker, A C., and A. W. Nagle, "Hardware/Software Tradeoffs in a Variable Word Width Queue Length Buffer Memory", Proceedings, 4th annual Symposium on Computer Architecture. IEEE, New York, 1977, pp. 159-163.
41. Parnas, D. L., "Information Distribution Aspects of Design Methodology", Information Processing 71 North-Holland Publishing Co., 1972.
42. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules", Communication of the ACM (Programming Techniques Department), December 1972.
43. Parnas, D. L., "A Technique for Software Module Specification with Examples", Communications of the ACM (Programming Techniques Department), May 1972.

44. Parnas, D. L., "Information Distribution Aspects of Design Methodology", Proceedings of IFIP Congress 1971.
45. Parnas, D. L., and W. R. Price., "The Design of the Virtual Memory Aspects of a Virtual Machine", Proceedings of the ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, March 1973.
46. Parnas, D. L., "The Use of Abstract Interfaces in the Development of Software for Embedded Computer Systems", Technical Memorandum of the Naval Research Laboratory, Washington, D.C. 20375.
47. Ramamborthy, C. V., and R. L. Kleir., "Optimization Strategies for Microprograms", IEEE Transactions on Computers, C-20 (July, 1971)
48. Roman, G. G., "Total System Methodology Framework". Summer, 1979.
49. Russell, R. D., "The PDP-11: A Case History of How Not To Design Condition Codes", Proc. of the 5th Annual Symposium on Architecture, 1978, pp. 190-194.
50. Yourdon, E., and Constantine, L., Structured Design, Prentice-Hall, Englewood Cliffs, N.Y., 1979.



*MISSION
of
Rome Air Development Center*

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

**DA
FILM**